Locality-Sensitive Hashing and Beyond

Ilya Razenshteyn (MIT)

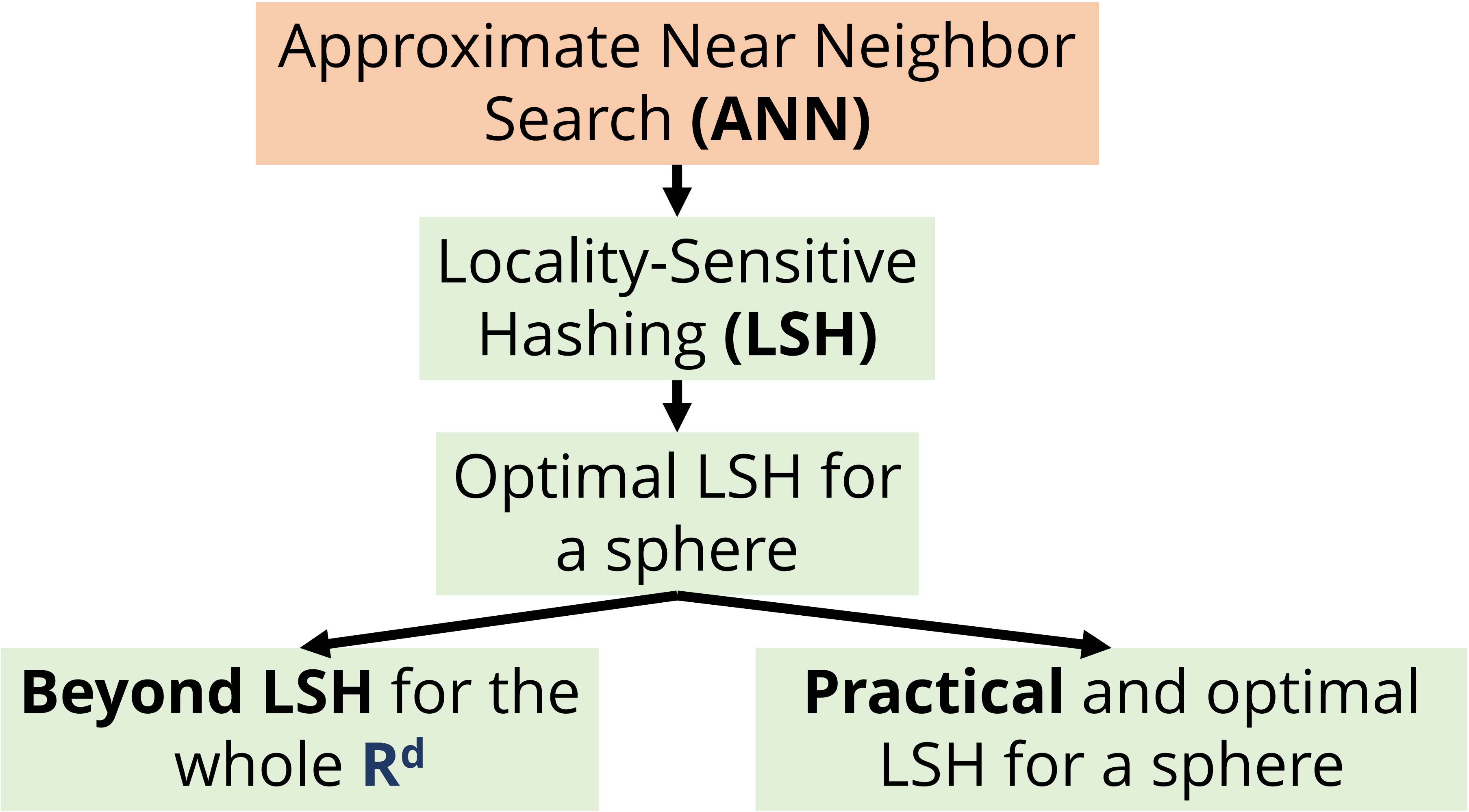
based on papers joint with

Alexandr Andoni (Columbia), Piotr Indyk (MIT),

Thijs Laarhoven (TU Eindhoven) and Ludwig Schmidt (MIT)

Outline

Outline


Approximate Near Neighbor Search **(ANN)**
Locality-Sensitive Hashing **(LSH)**
Optimal LSH for a sphere
Beyond LSH for the whole $\mathbf{R}^d$
Practical and optimal LSH for a sphere

Outline

Approximate Near Neighbor Search **(ANN)**

↓

Locality-Sensitive Hashing **(LSH)**

↓

Optimal LSH for a sphere

[Andoni, R 2015] STOC 2015

[Andoni, Indyk, Laarhoven, R, Schmidt 2015] NIPS 2015

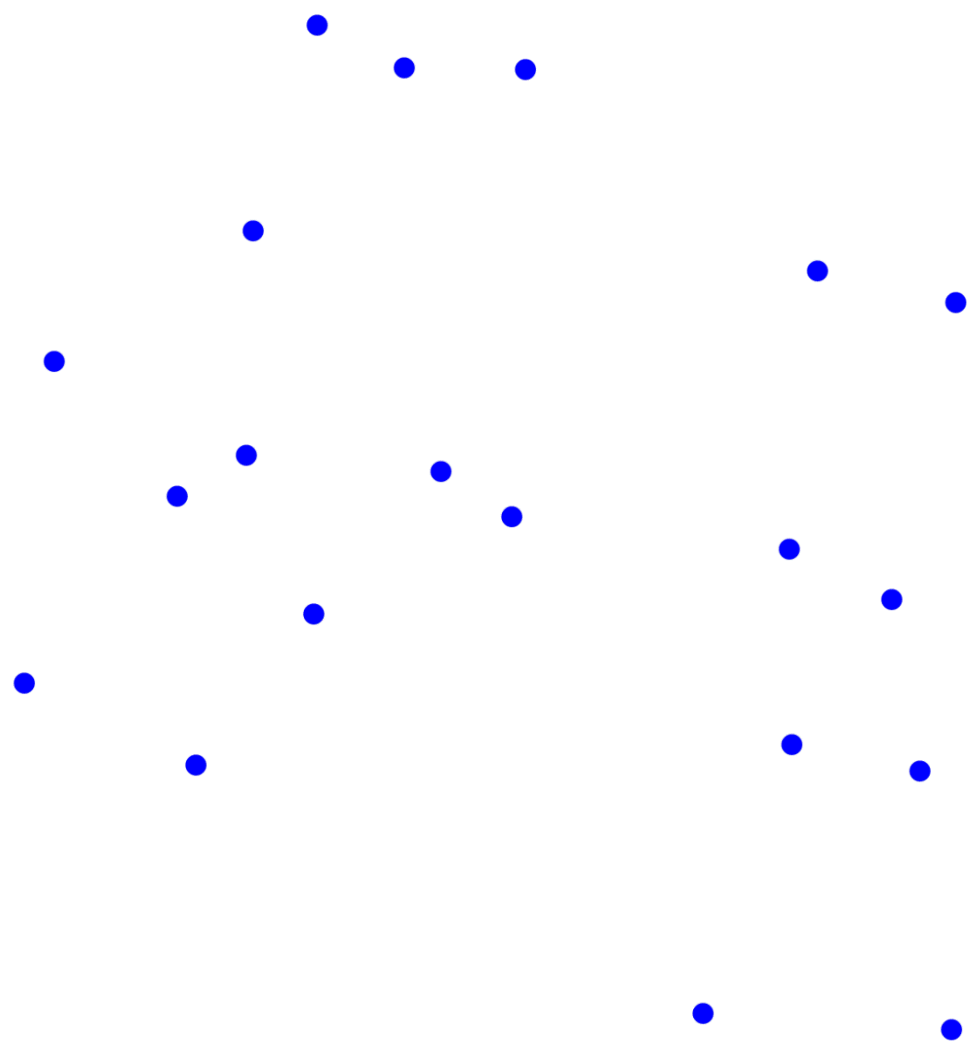
Beyond LSH for the whole $\mathbf{R}^d$

Practical and optimal LSH for a sphere

Near Neighbor Search

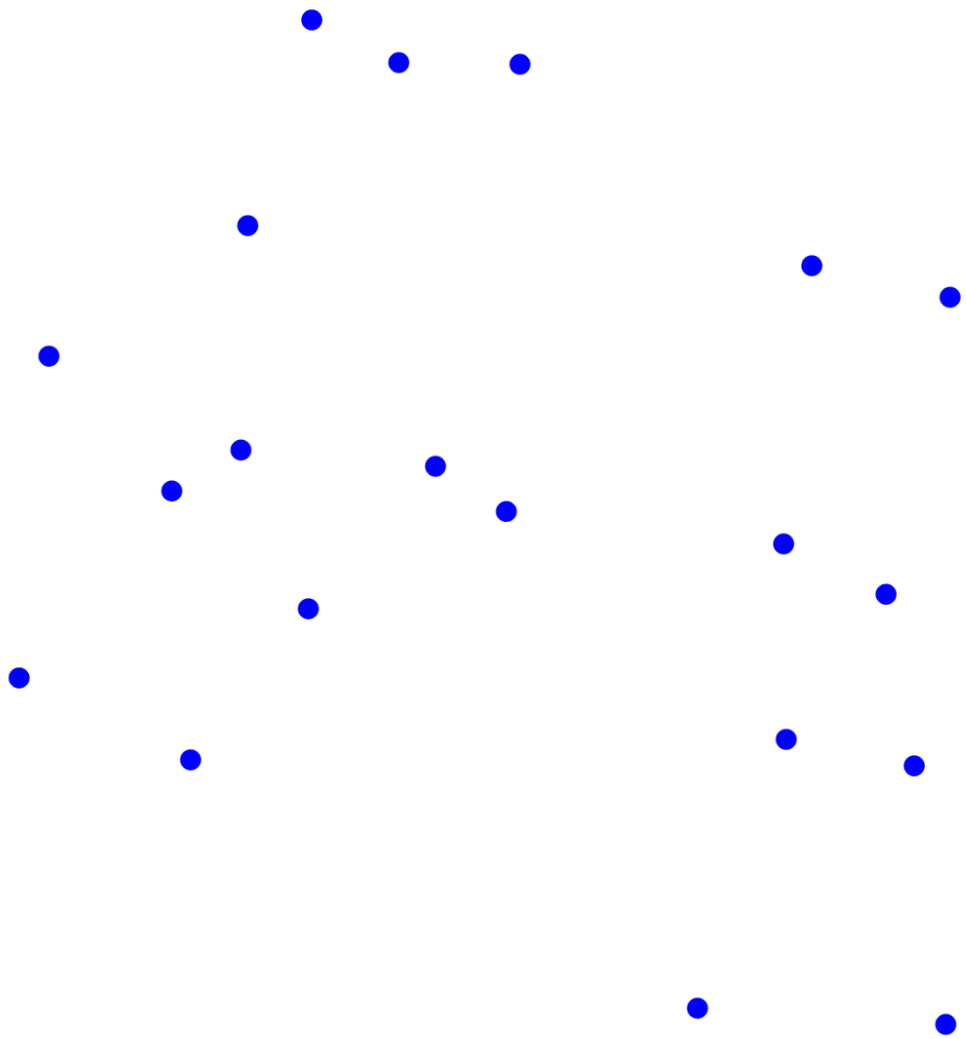
Near Neighbor Search

- **Dataset:** $\mathbf{n}$ points in $\mathbf{R^d}$, $\mathbf{r > 0}$



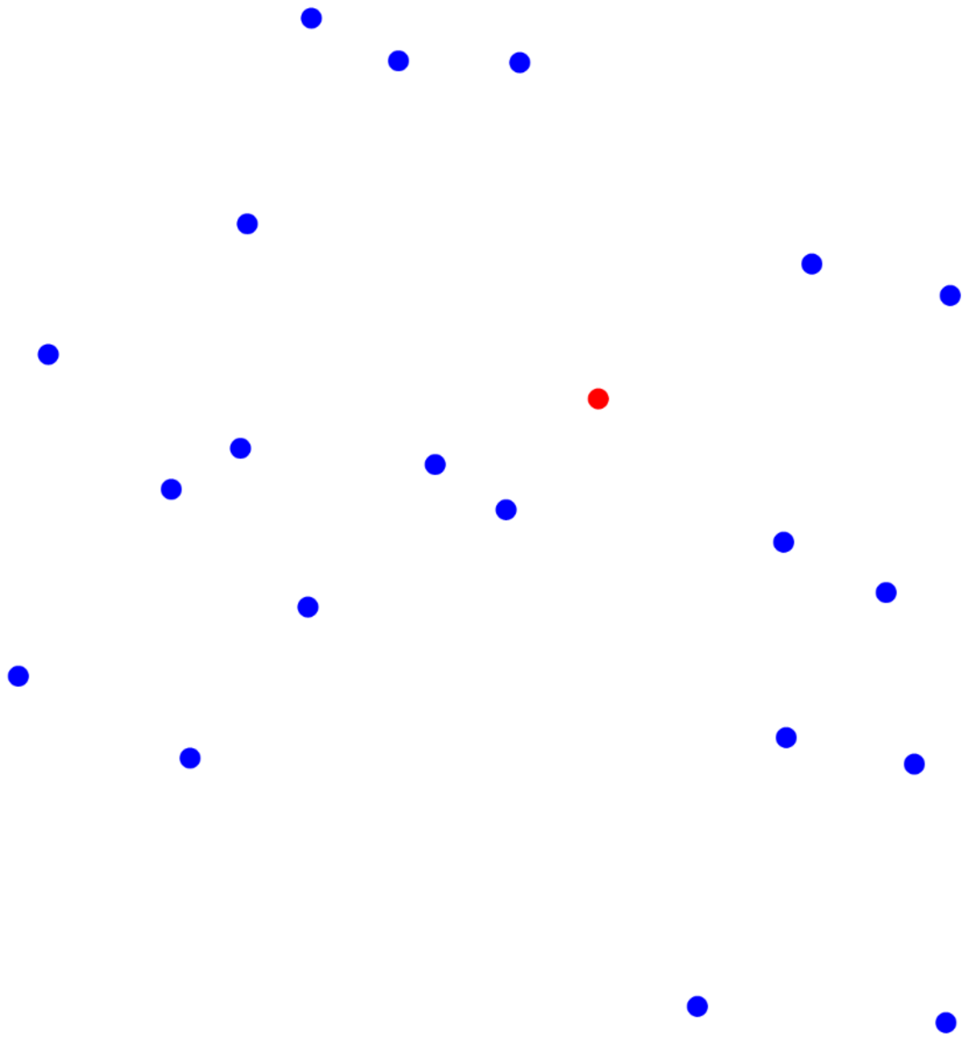
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query



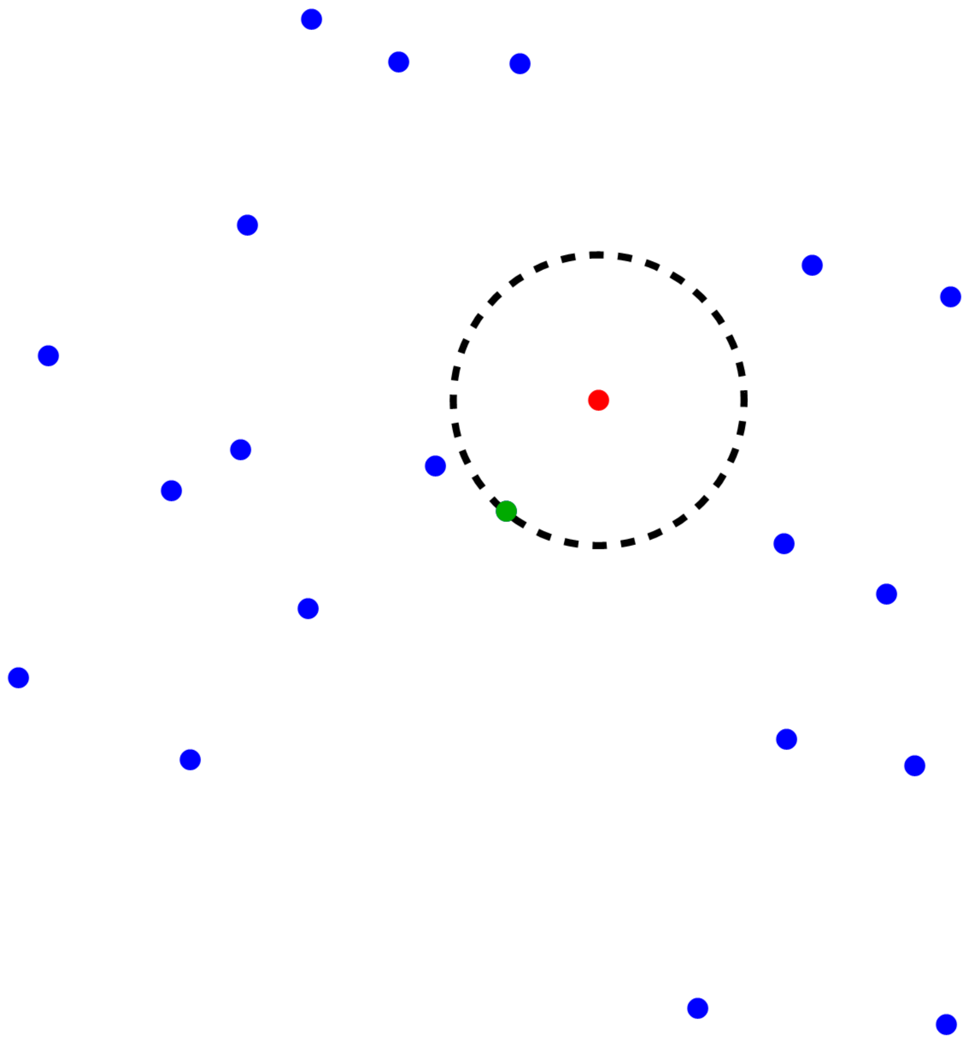
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, $\mathbf{r > 0}$
- **Goal:** a data point within **r** from a query



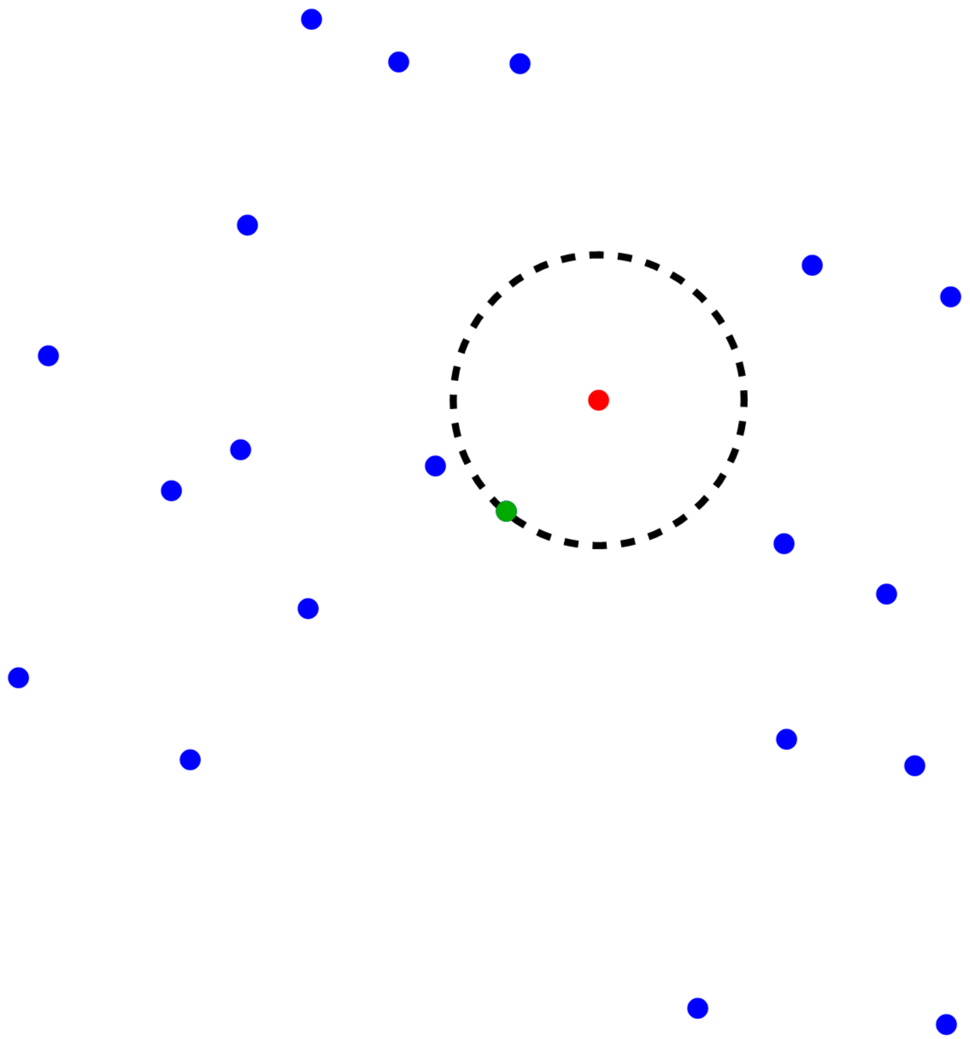
Near Neighbor Search

- **Dataset:** **n** points in **R^d**, **$r > 0$**
- **Goal:** a data point within **r** from a query



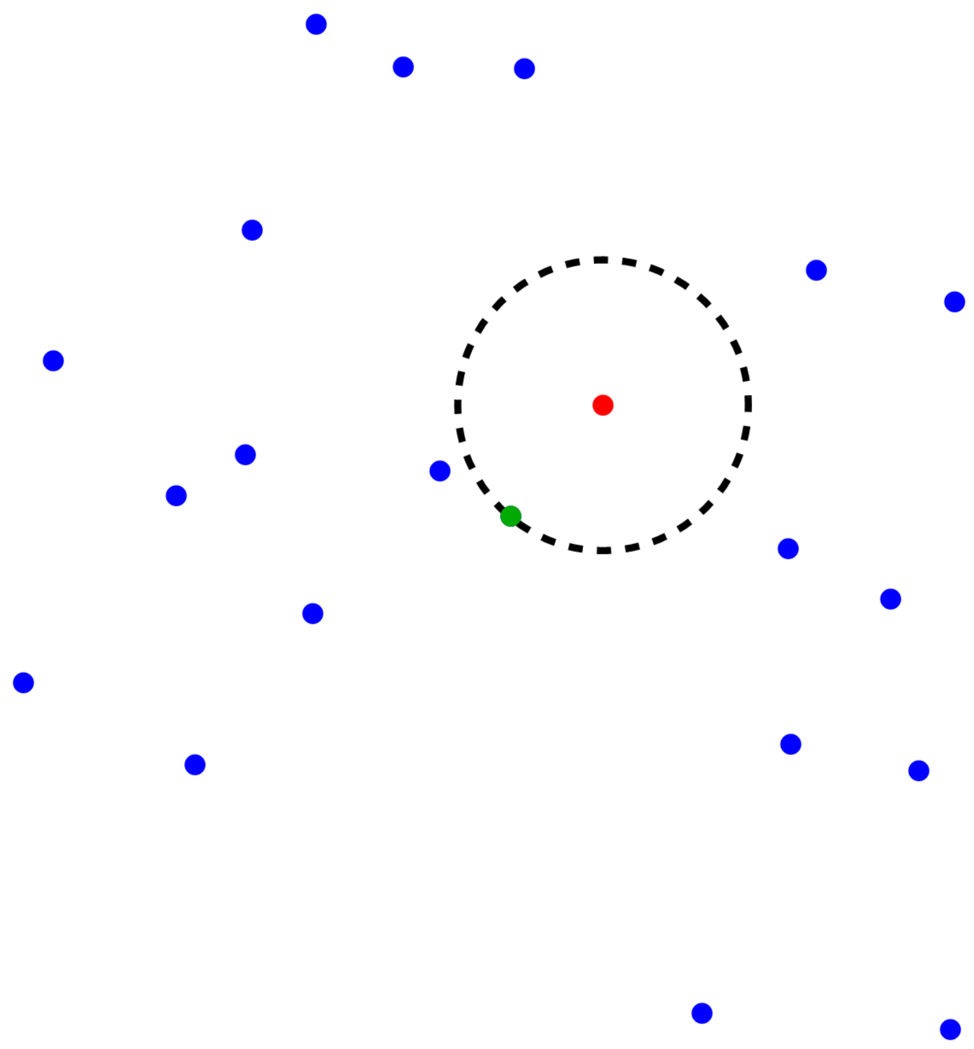
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, $\mathbf{r > 0}$
- **Goal:** a data point within **r** from a query
- Space, query time



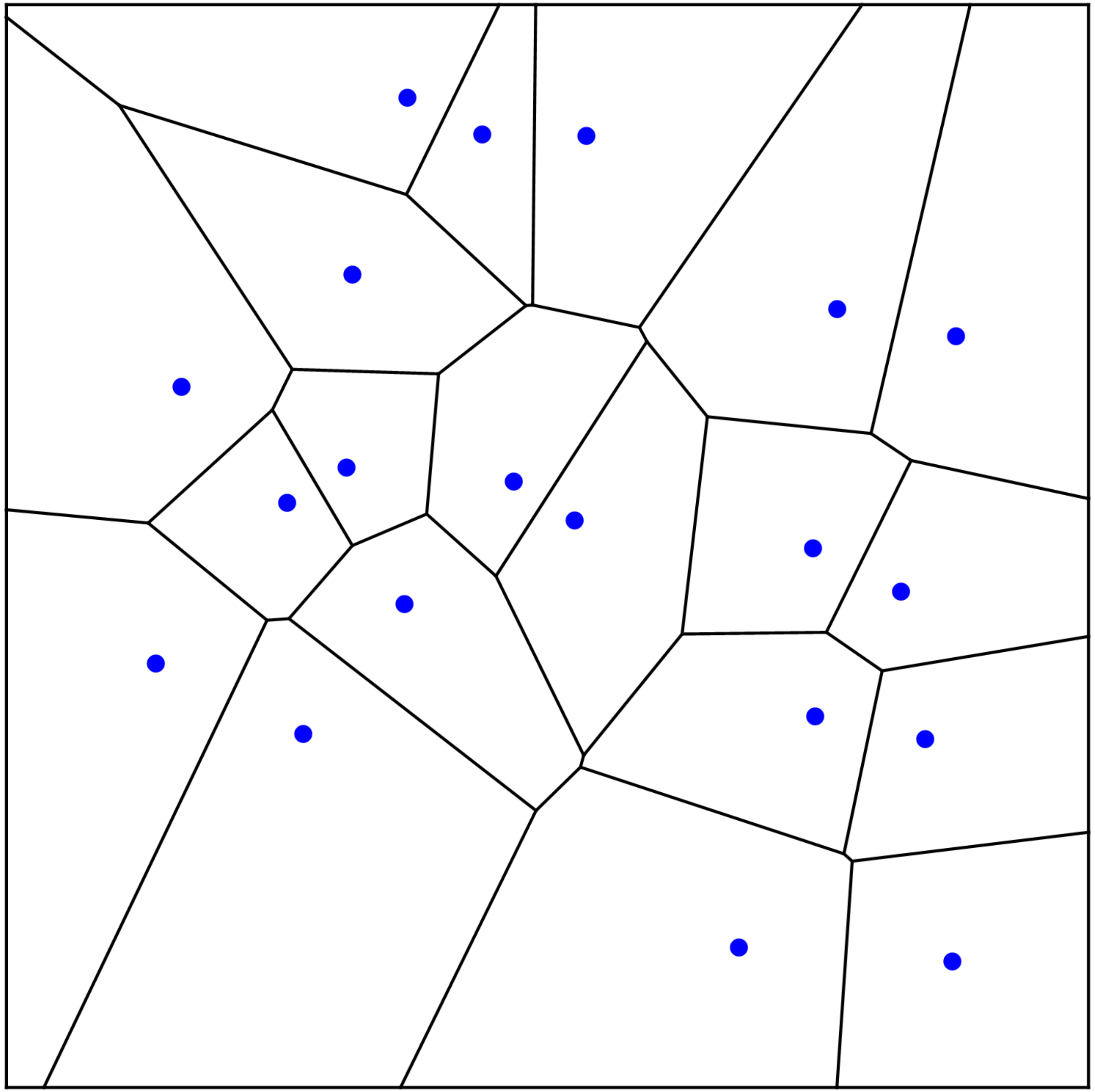
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



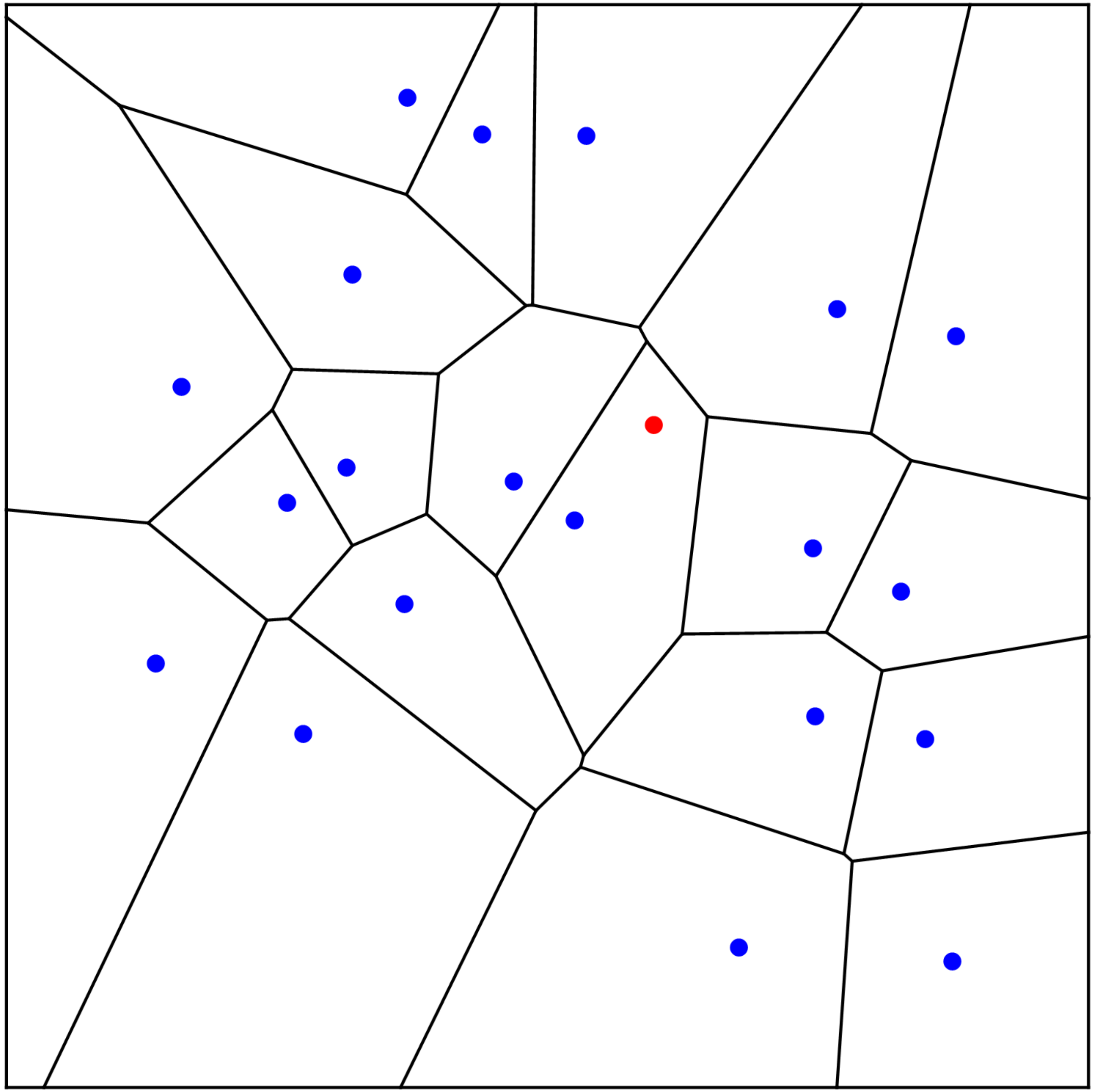
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



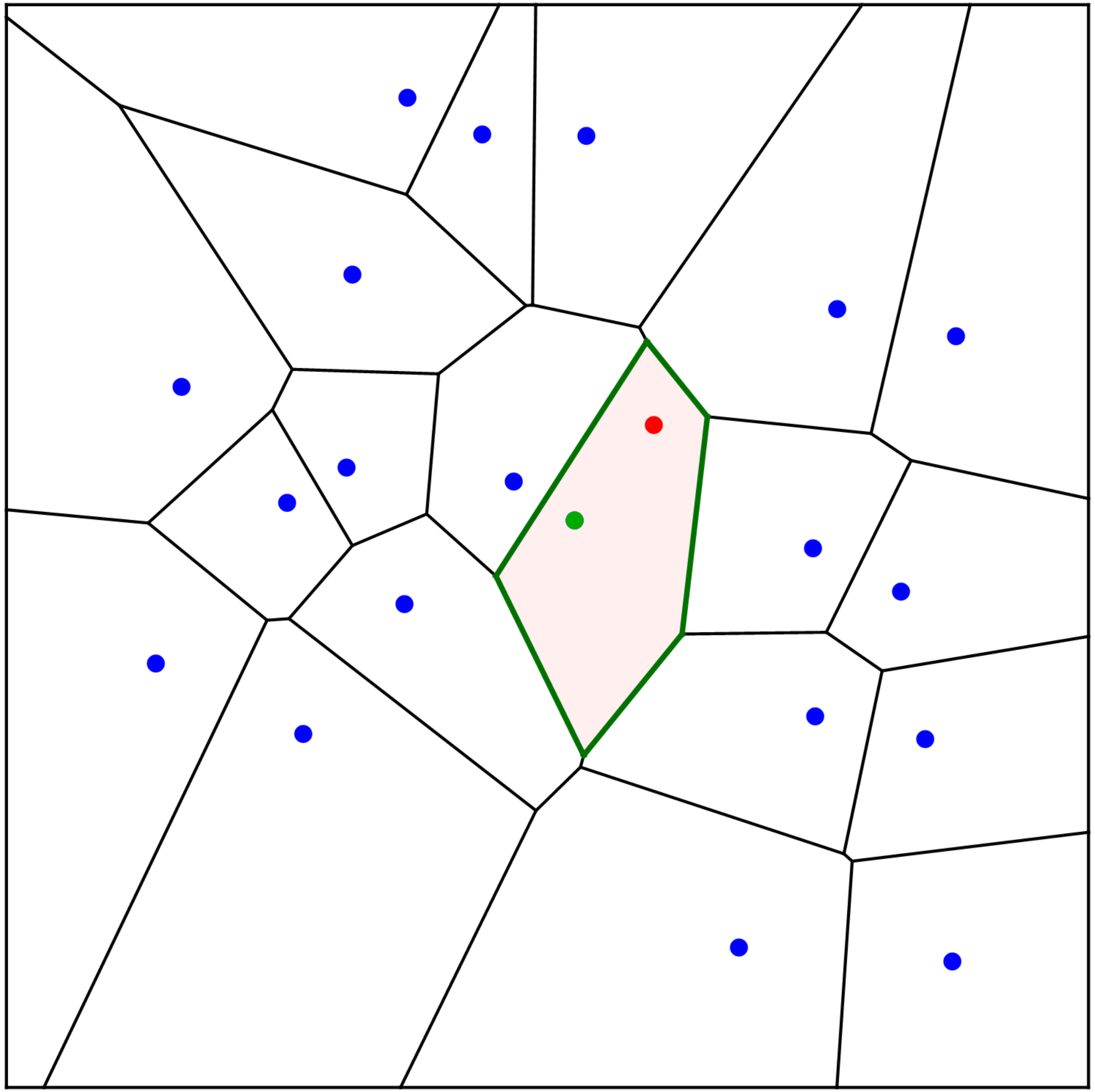
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



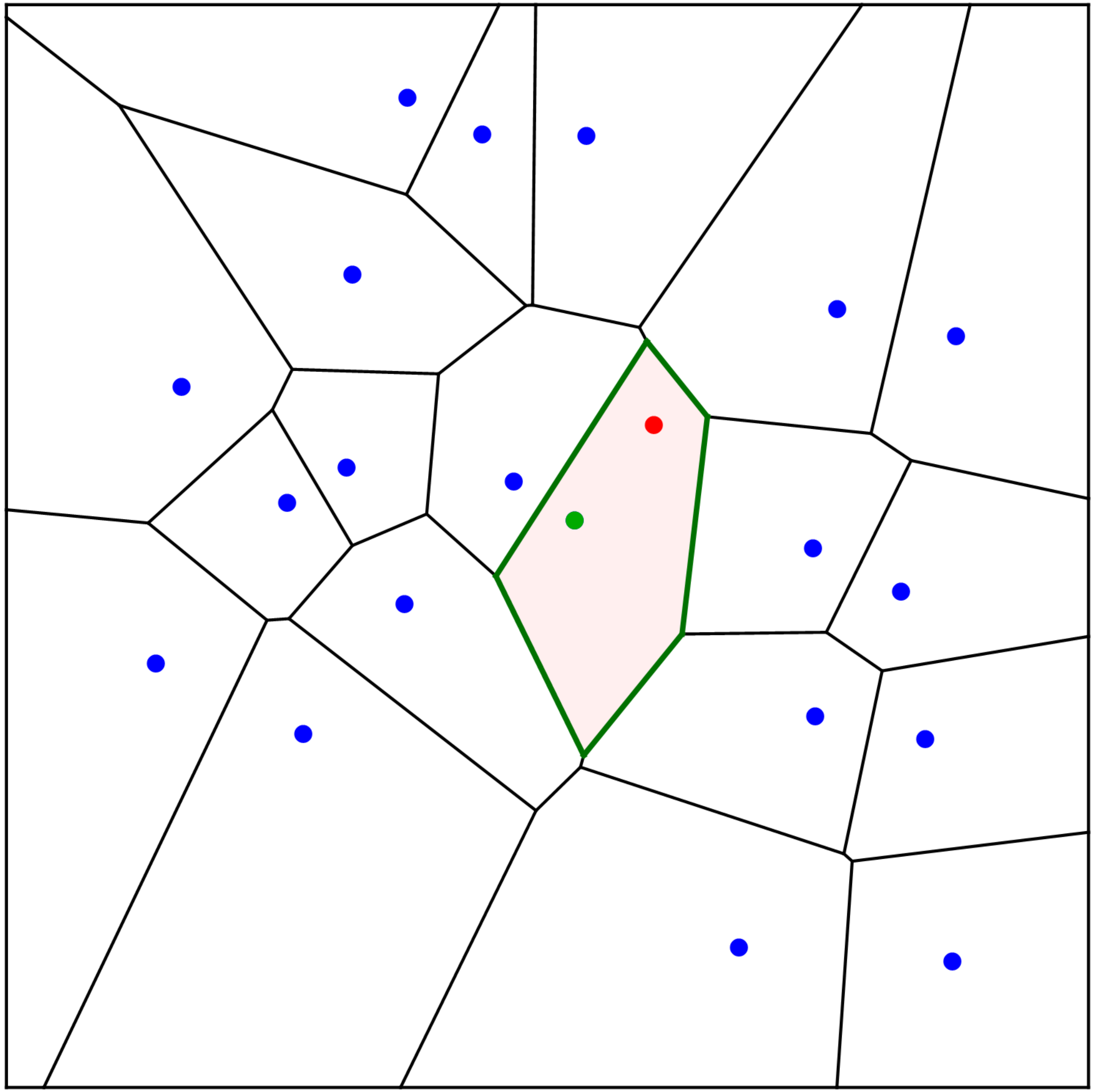
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



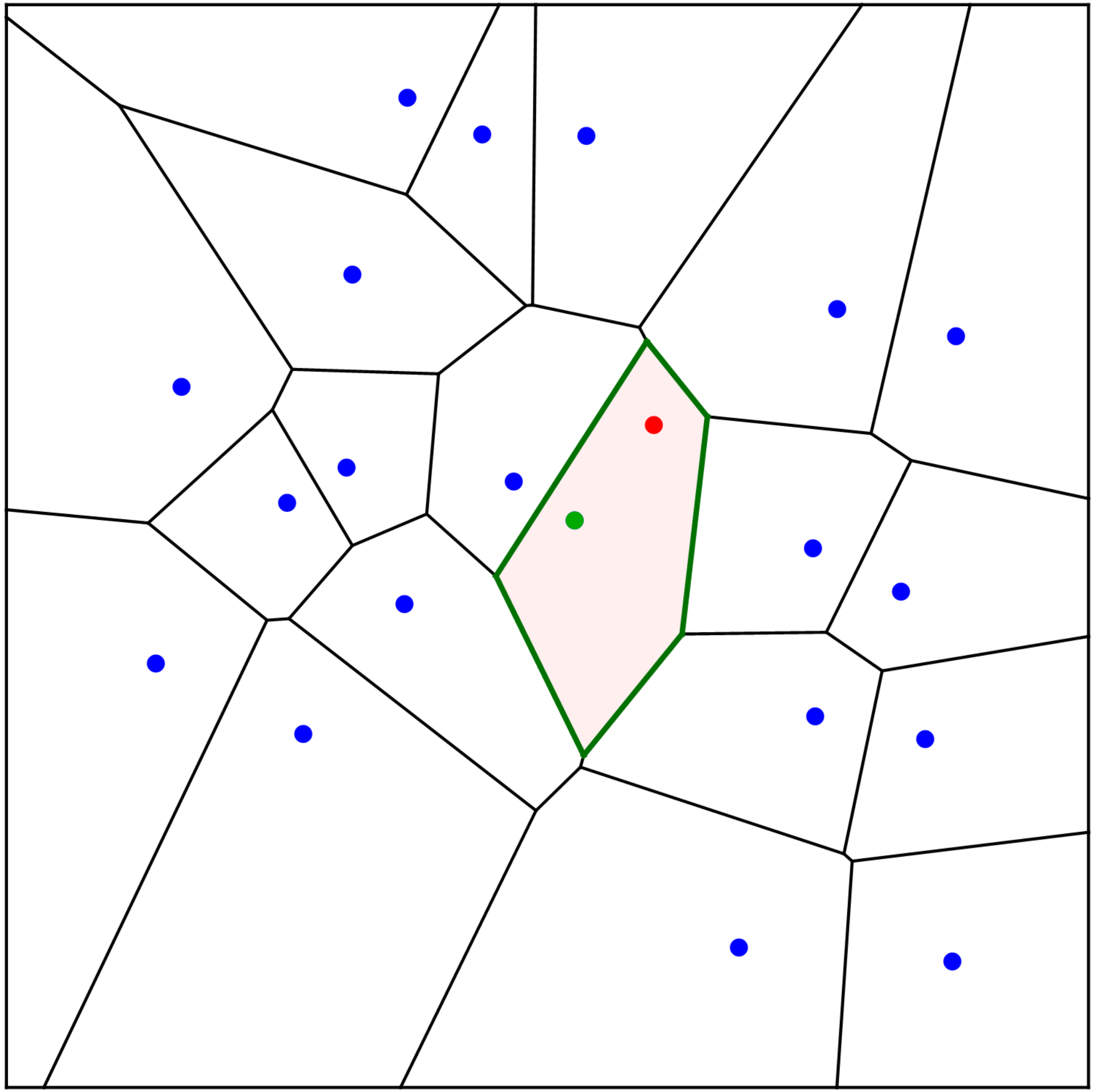
Near Neighbor Search

- **Dataset:** **n** points in **R^d**, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time
- Infeasible for large **d**:
  - Space exponential in the dimension



Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time
- Infeasible for large **d**:
  - Space exponential in the dimension
- Most of the applications are in high dimensions



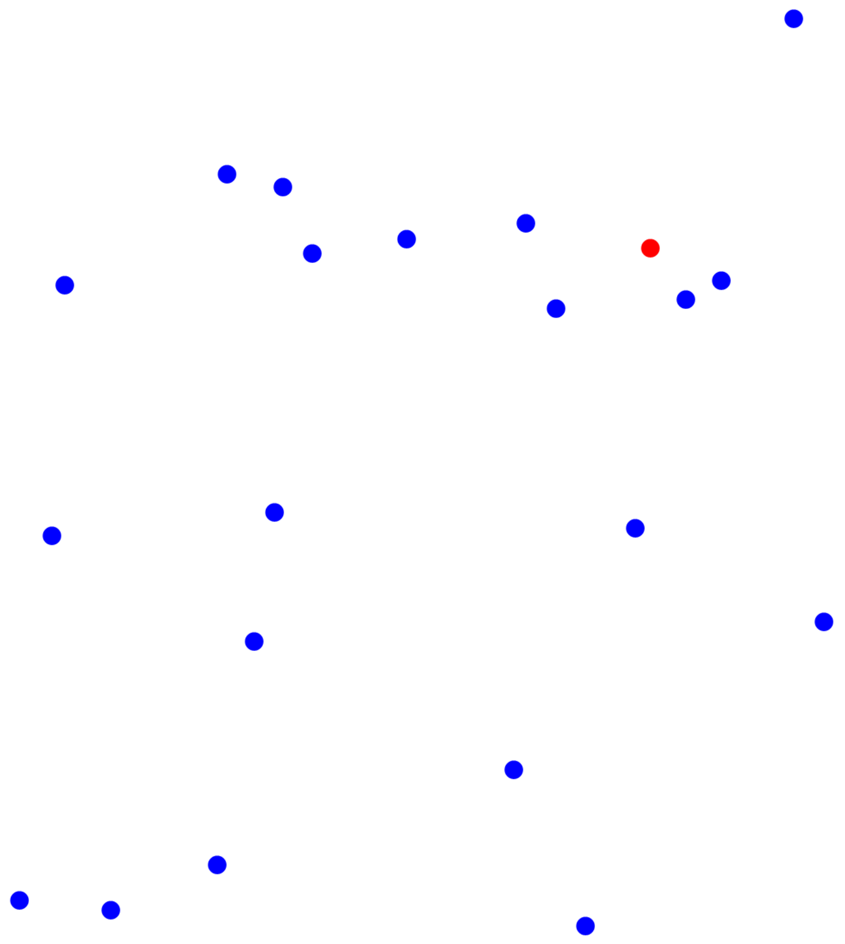
Approximate Near Neighbor Search (ANN)

Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in **R^d**
  - distance threshold **$r > 0$**
  - approximation **$c > 1$**

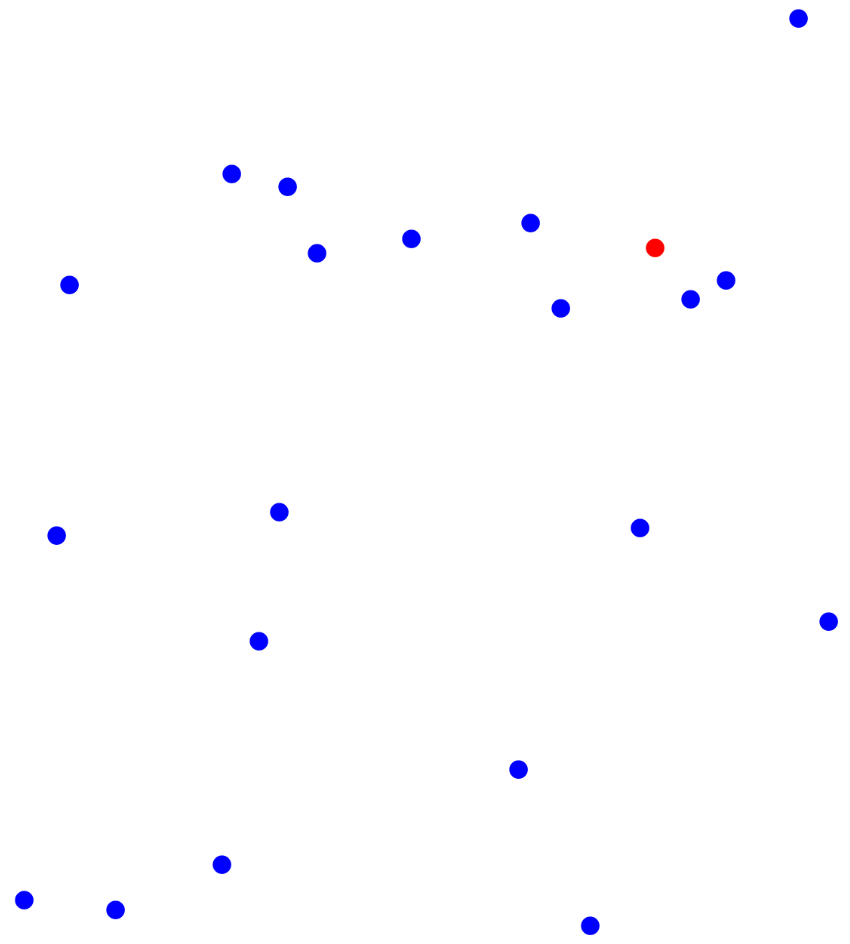
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$



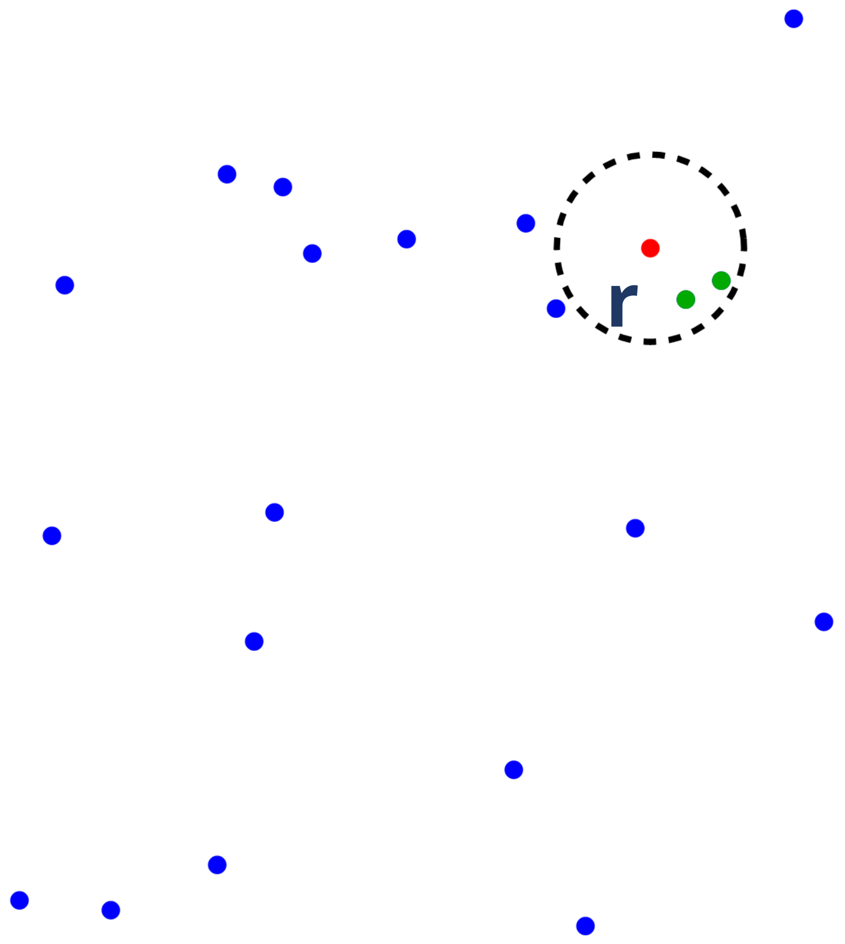
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in **R^d**
  - distance threshold **$r > 0$**
  - approximation **$c > 1$**
- **Query:** a point within **r** from a data point



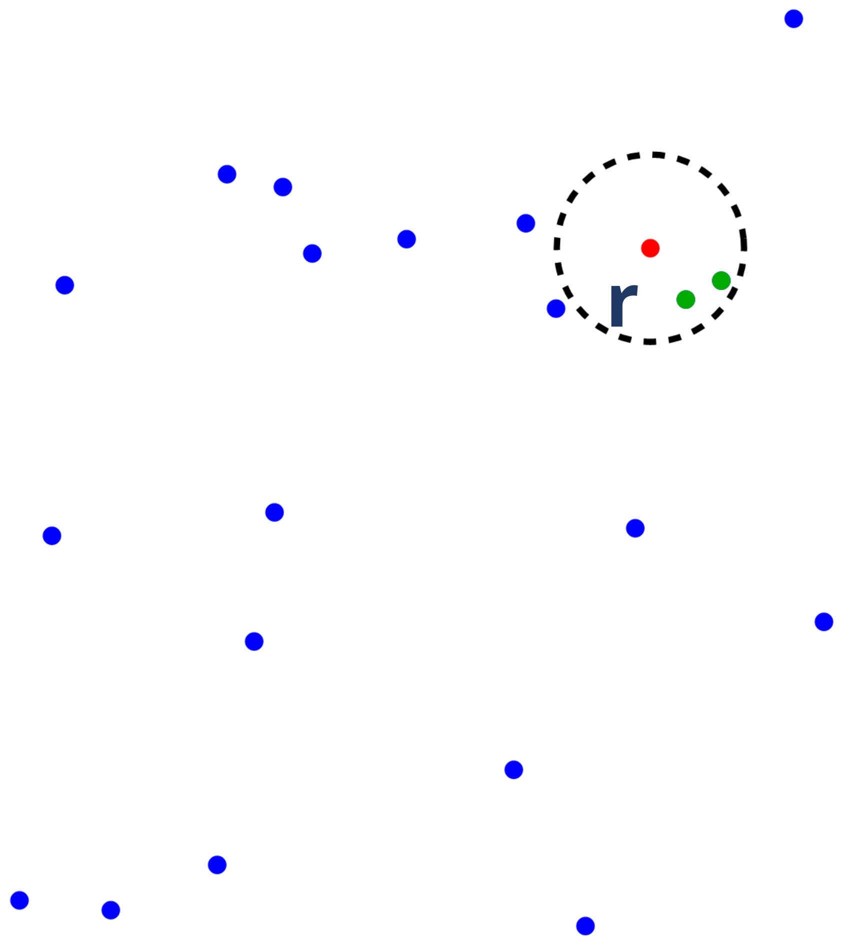
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold **r > 0**
  - approximation **c > 1**
- **Query:** a point within **r** from a data point



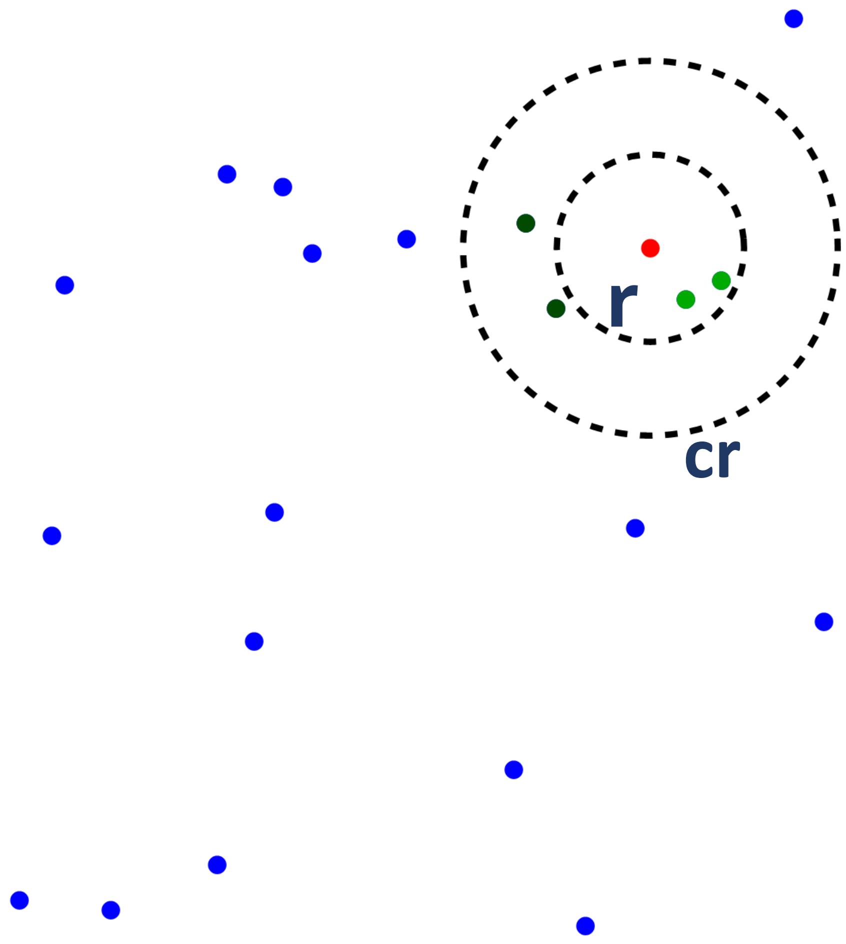
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold **r > 0**
  - approximation **c > 1**
- **Query:** a point within **r** from a data point
- **Want:** a data point within **cr** from the query



Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold **r > 0**
  - approximation **c > 1**
- **Query:** a point within **r** from a data point
- **Want:** a data point within **cr** from the query



Applications

Applications

- Similarity search for: images, audio, video, texts, biological data etc

Applications

- Similarity search for: images, audio, video, texts, biological data etc
- Cryptanalysis (the Shortest Vector Problem in lattices) **[Laarhoven 2015]**

Applications

- Similarity search for: images, audio, video, texts, biological data etc
- Cryptanalysis (the Shortest Vector Problem in lattices) **[Laarhoven 2015]**
- Optimization: Coordinate Descent **[Dhillon, Ravikumar, Tewari 2011]**, Stochastic Gradient Descent **[Hofmann, Lucchi, McWilliams 2015]** etc

Spherical case

Spherical case

- Very important case:

    all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

Spherical case

- Very important case:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?

Spherical case

- Very important case:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?
- **In theory:** can reduce general case to the spherical case (later in the talk)

Spherical case

- Very important case:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?
- **In theory:** can reduce general case to the spherical case (later in the talk)
- **In practice:**
  - Cosine similarity is widely used
  - Oftentimes, can boldly pretend that the dataset lies on a sphere and *be just fine*

Spherical *random* case

Spherical *random* case

- **Dataset:** **n** random points on a sphere

Spherical *random* case

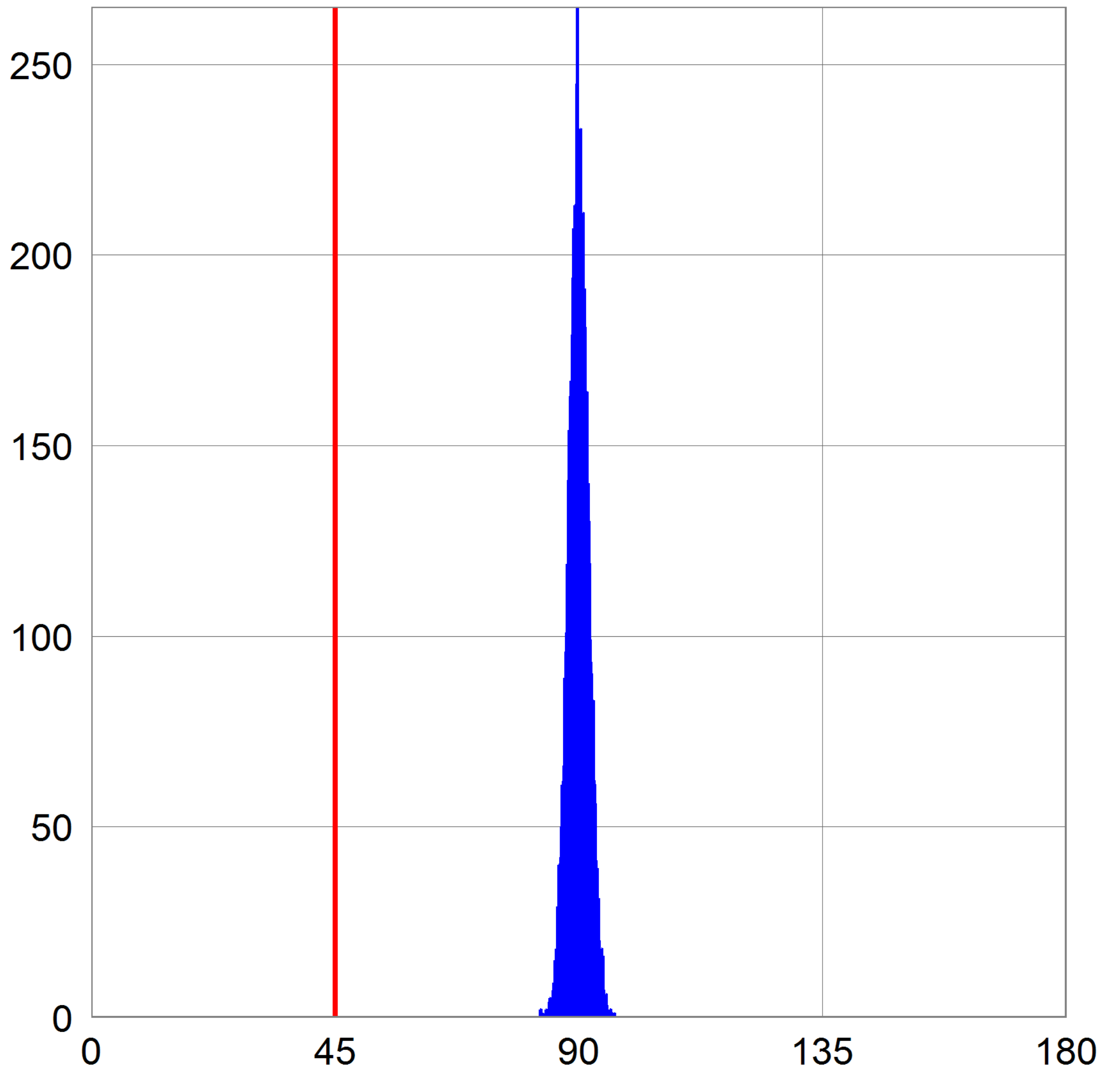
- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point

Spherical *random* case

- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!

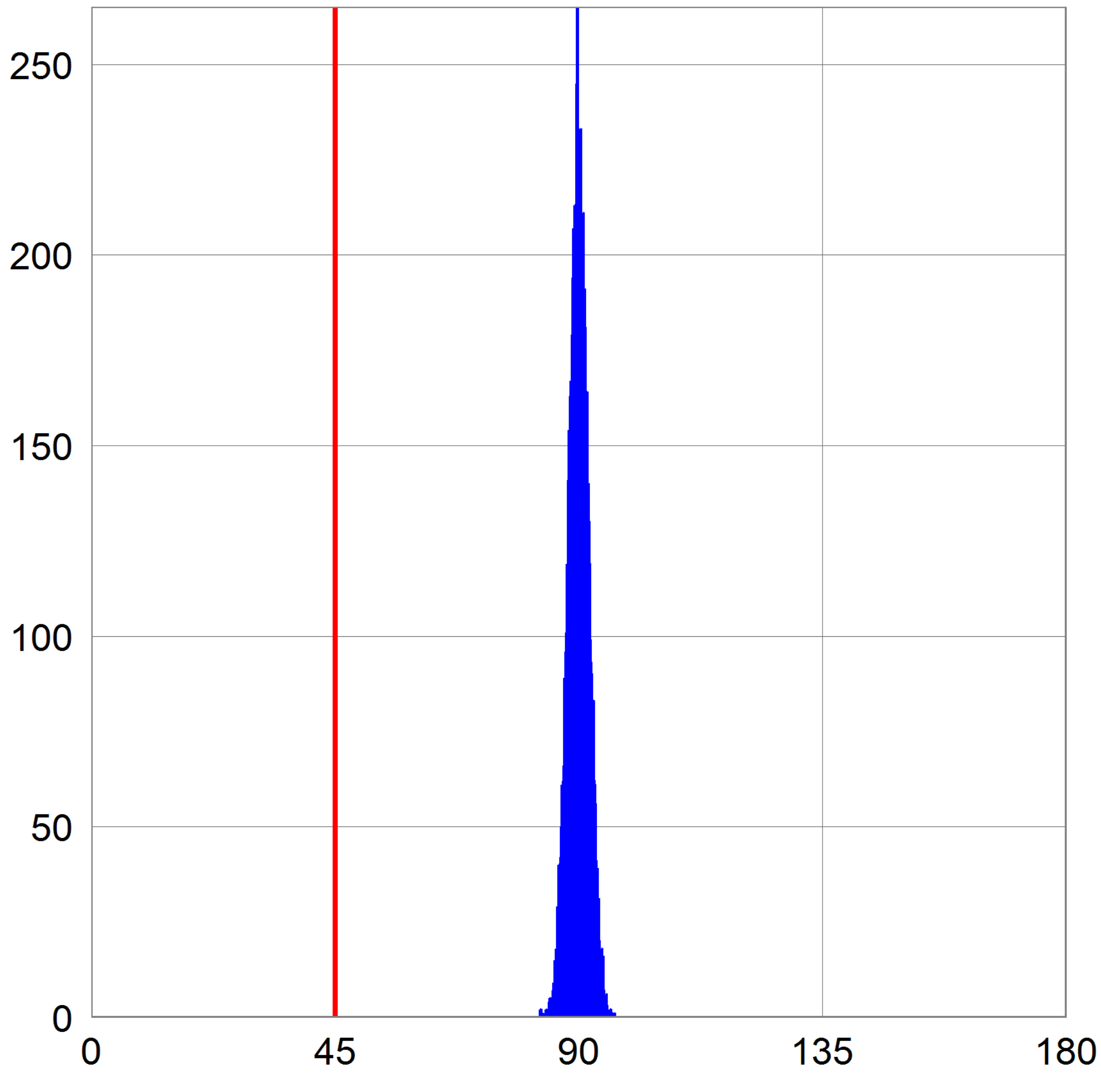
Spherical *random* case

- **Dataset: n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!



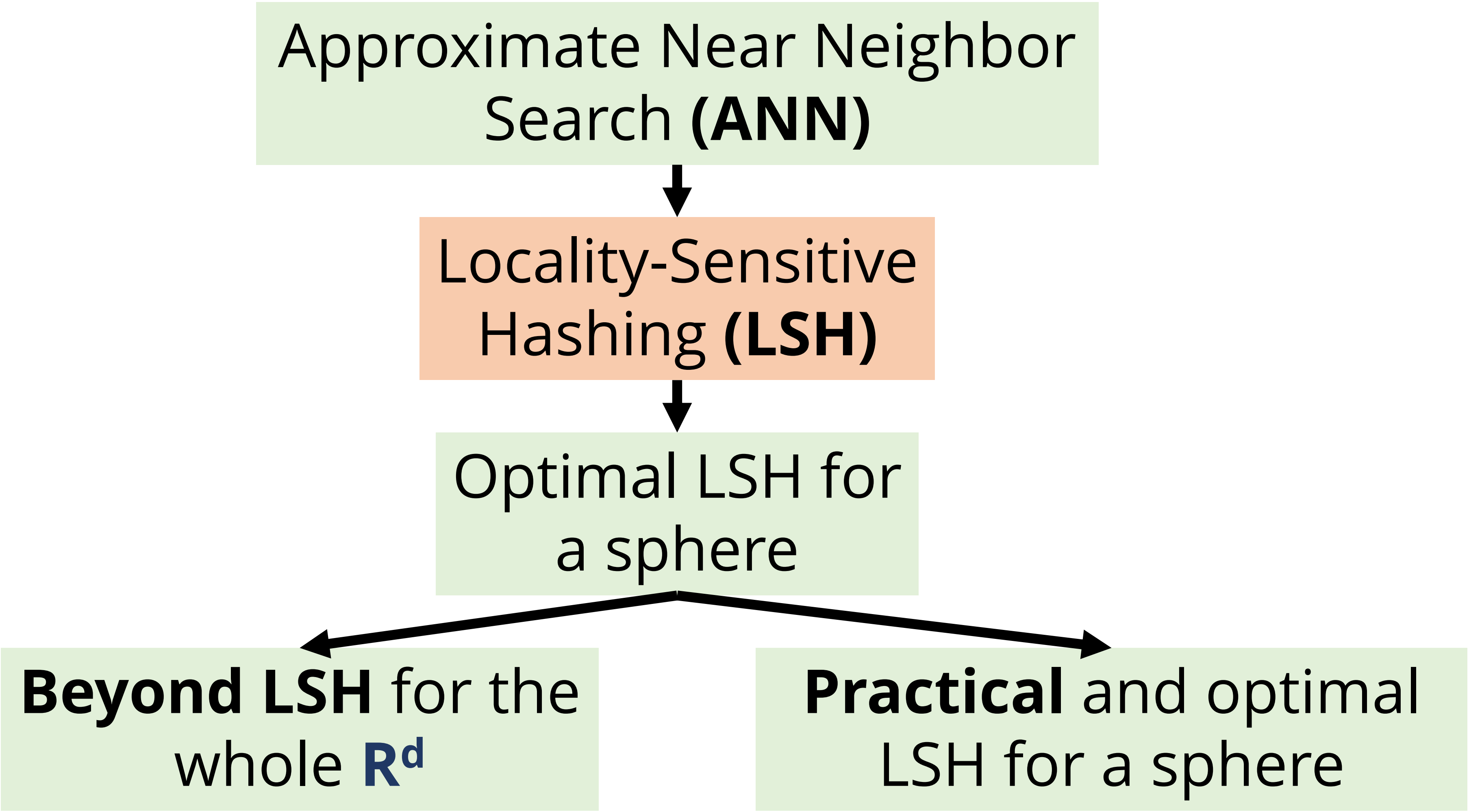
Spherical *random* case

- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!
- Instructive case to think about
  - Concentration of angles around **90** degrees happens in practice



Outline

Outline


Approximate Near Neighbor Search **(ANN)**
Locality-Sensitive Hashing **(LSH)**
Optimal LSH for a sphere
Beyond LSH for the whole **R^d**
Practical and optimal LSH for a sphere

Locality-Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition $\mathbf{R}$ is $(\mathbf{r}, \mathbf{cr}, \mathbf{p_1}, \mathbf{p_2})$-**sensitive** if for every $\mathbf{p}, \mathbf{q}$:
  - If $\|p - q\| \le r$, then $\Pr_R[R(p) = R(q)] \ge p_1$
  - If $\|p - q\| \ge cr$, then $\Pr_R[R(p) = R(q)] \le p_2$

Locality-Sensitive Hashing (LSH)

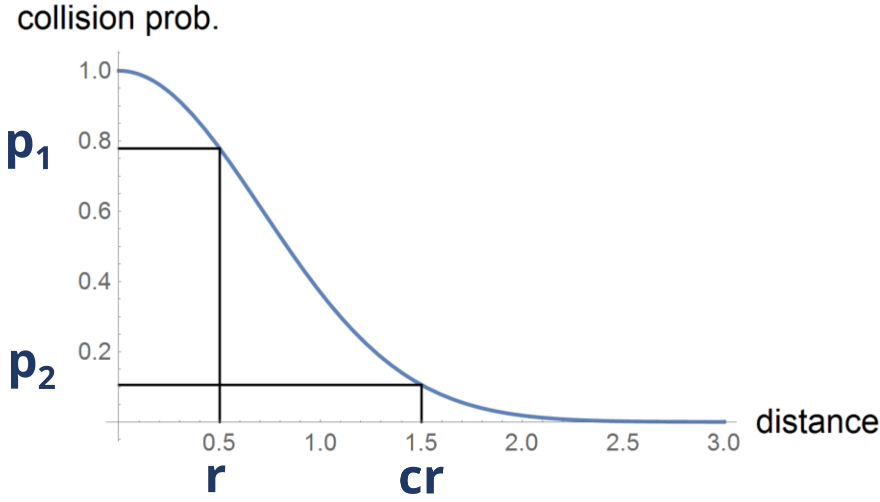
- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition **R** is (r, cr, p_1, p_2)-**sensitive** if for every **p, q**:
  - If $\|p - q\| \leq r$, then $\Pr_R[R(p) = R(q)] \geq p_1$
  - If $\|p - q\| \geq cr$, then $\Pr_R[R(p) = R(q)] \leq p_2$

From the definition of ANN

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition $\mathbf{R}$ is $\mathbf{(r, cr, p_1, p_2)}$**-sensitive** if for every $\mathbf{p, q}$:
  - If $\|\mathbf{p} - \mathbf{q}\| \le \mathbf{r}$, then $\mathbf{Pr_R[R(p) = R(q)] \ge p_1}$
  - If $\|\mathbf{p} - \mathbf{q}\| \ge \mathbf{cr}$, then $\mathbf{Pr_R[R(p) = R(q)] \le p_2}$

From the definition of ANN



Hyperplane LSH

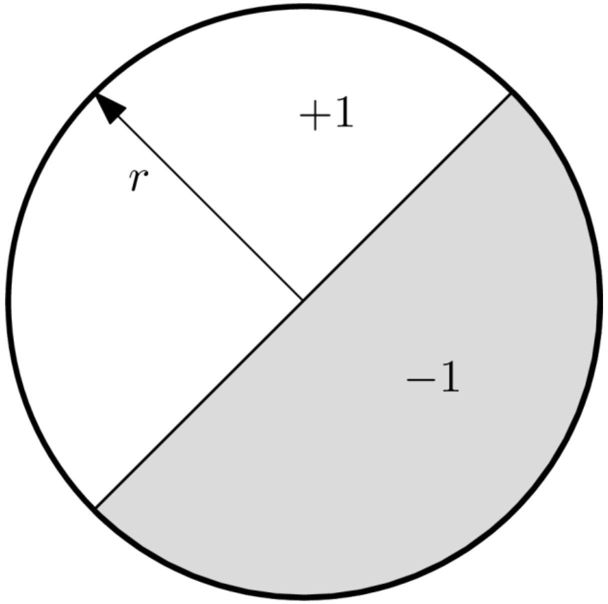
Hyperplane LSH

- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**

Hyperplane LSH

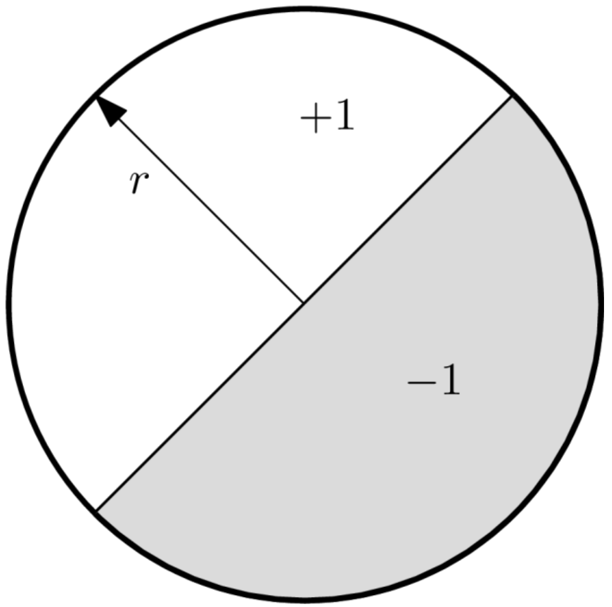
- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**

Hyperplane LSH



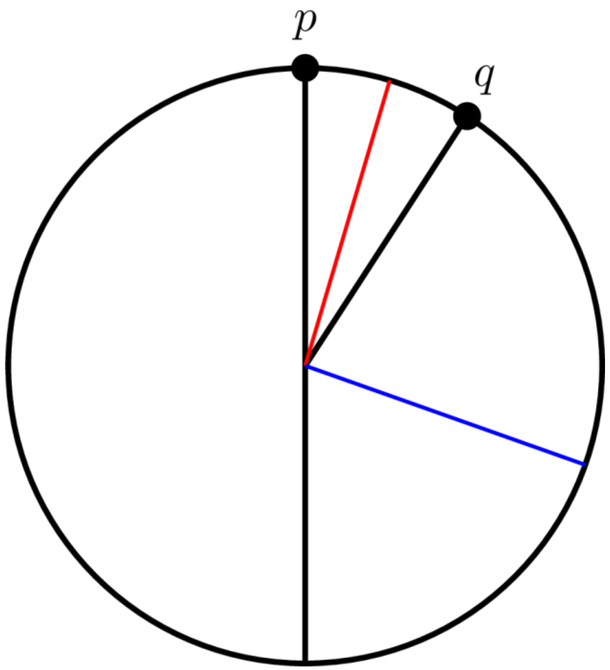
- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**

Hyperplane LSH



- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**

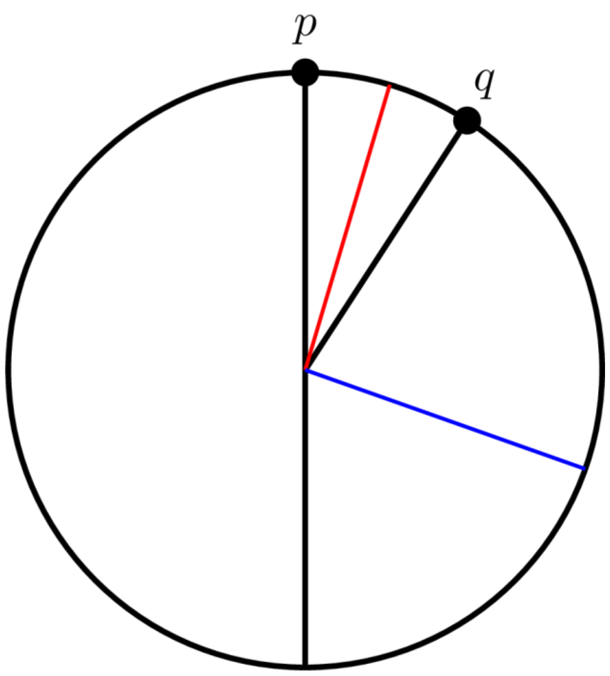
Hyperplane LSH

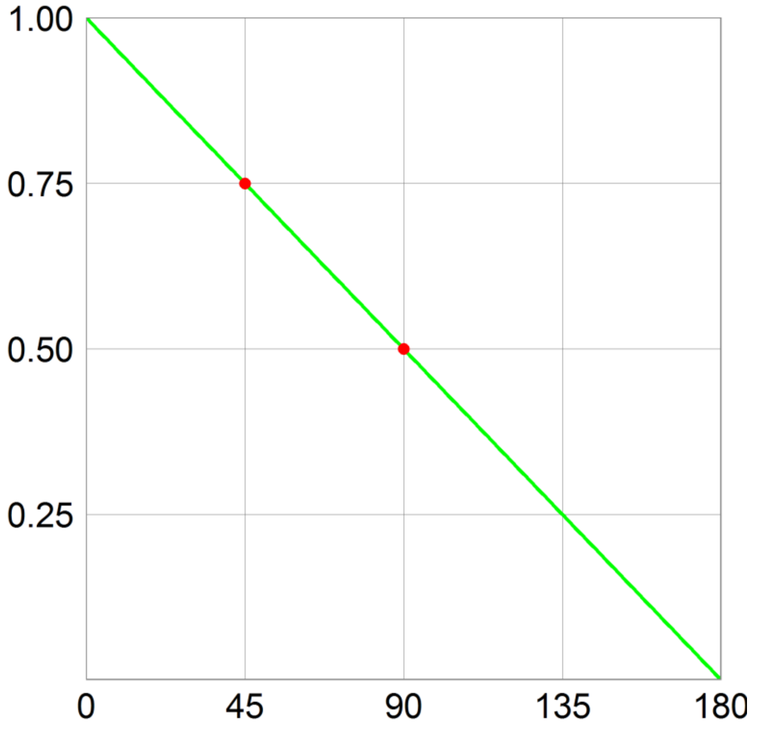


- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**

Hyperplane LSH

- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**





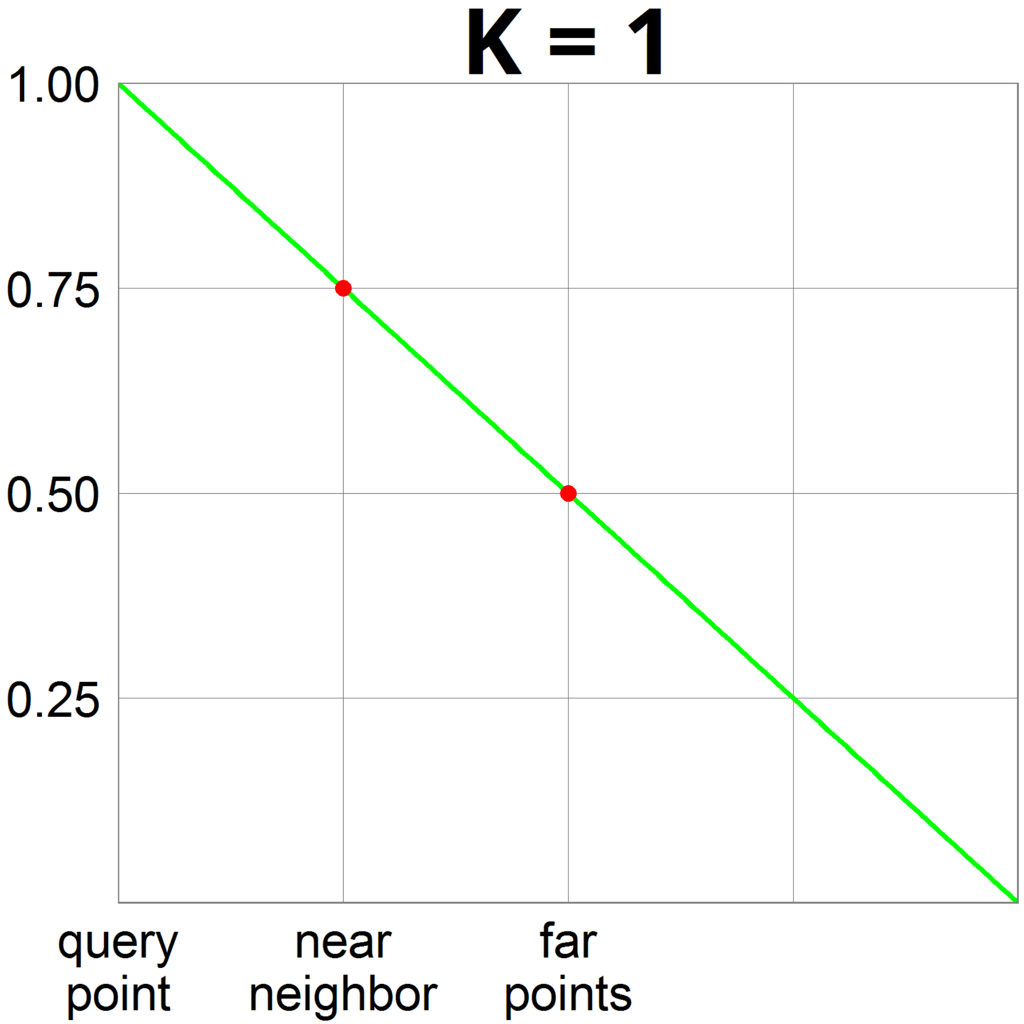
Using LSH to solve ANN

Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)

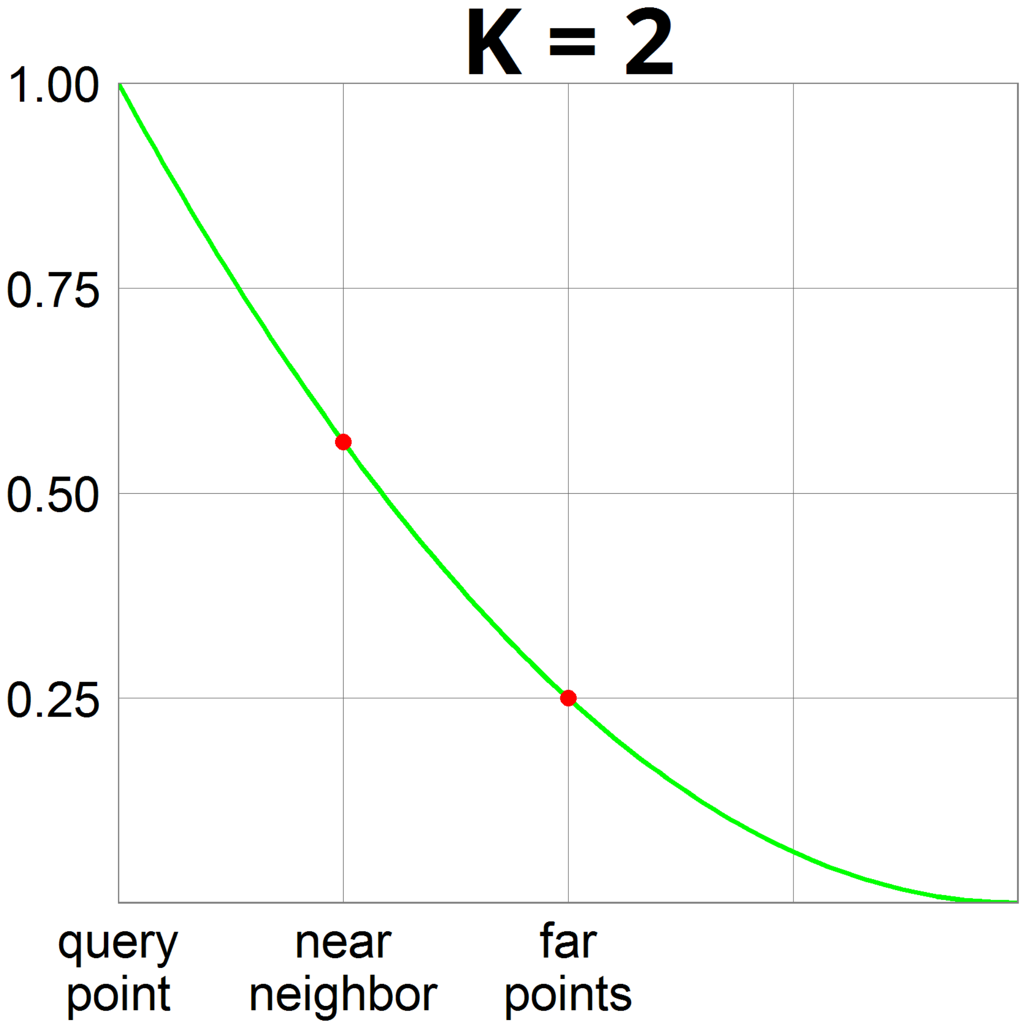
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)



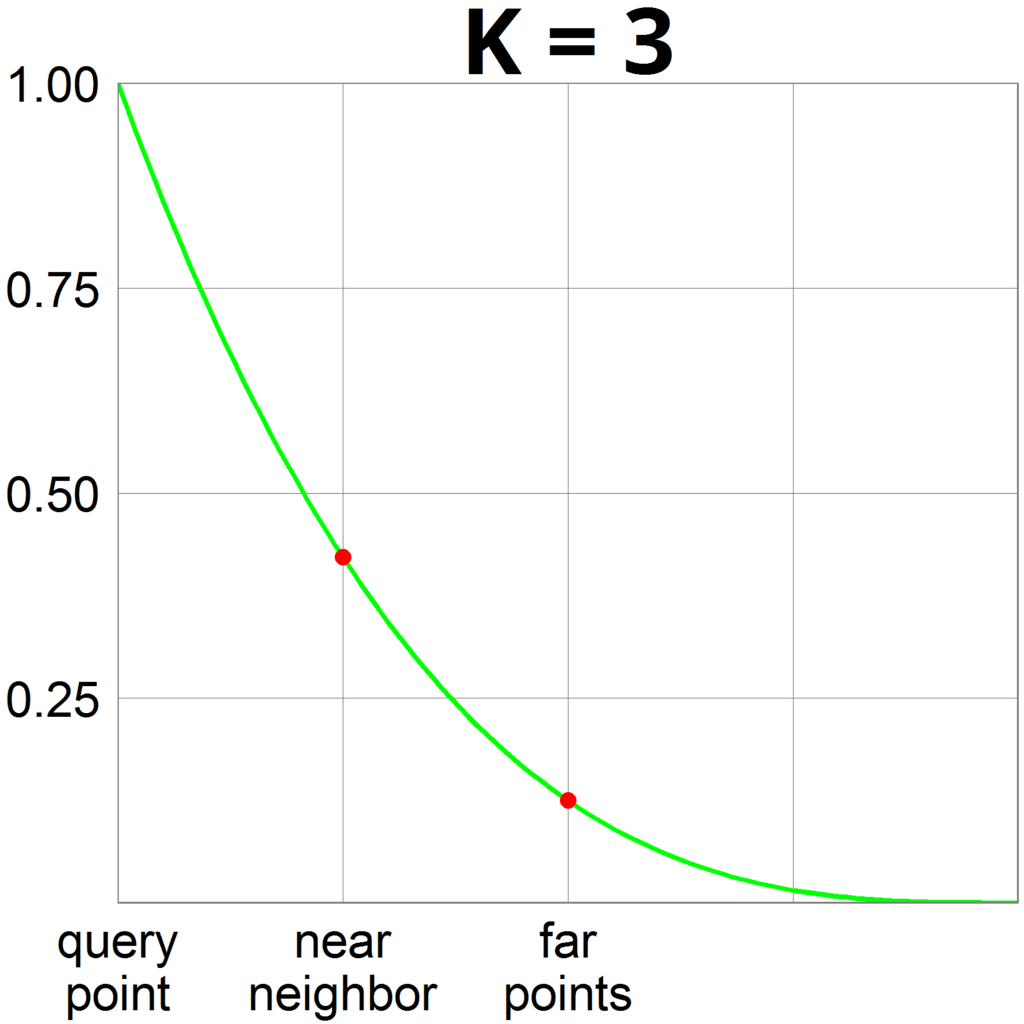
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)



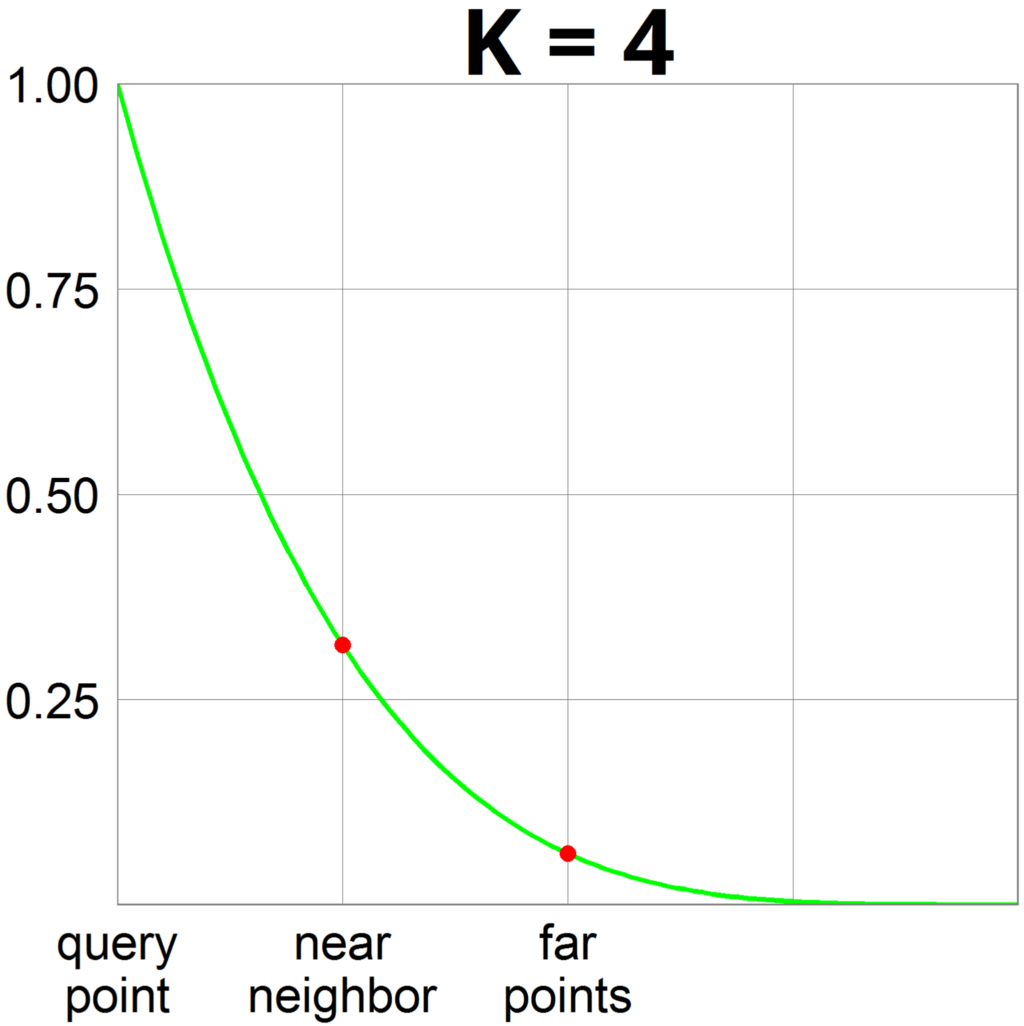
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p)))**



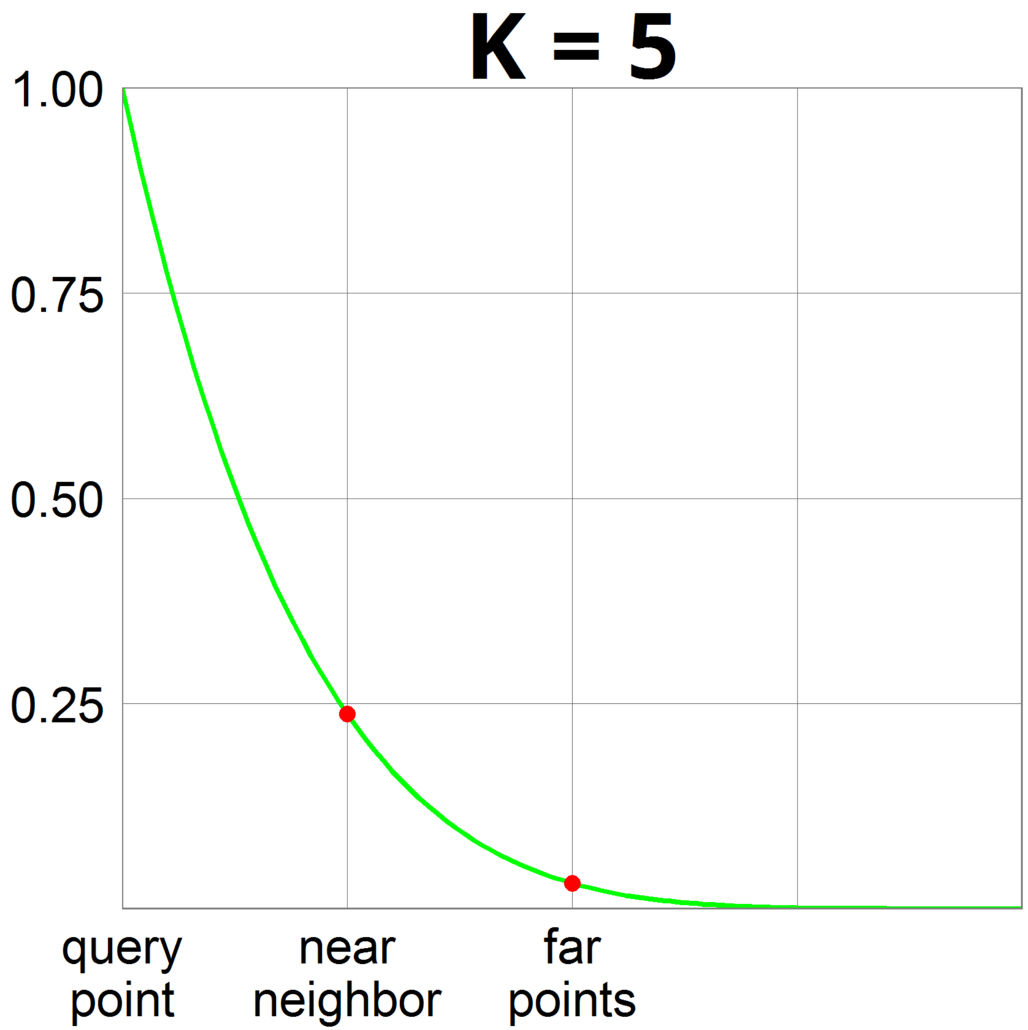
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), ..., h_K(p))**)



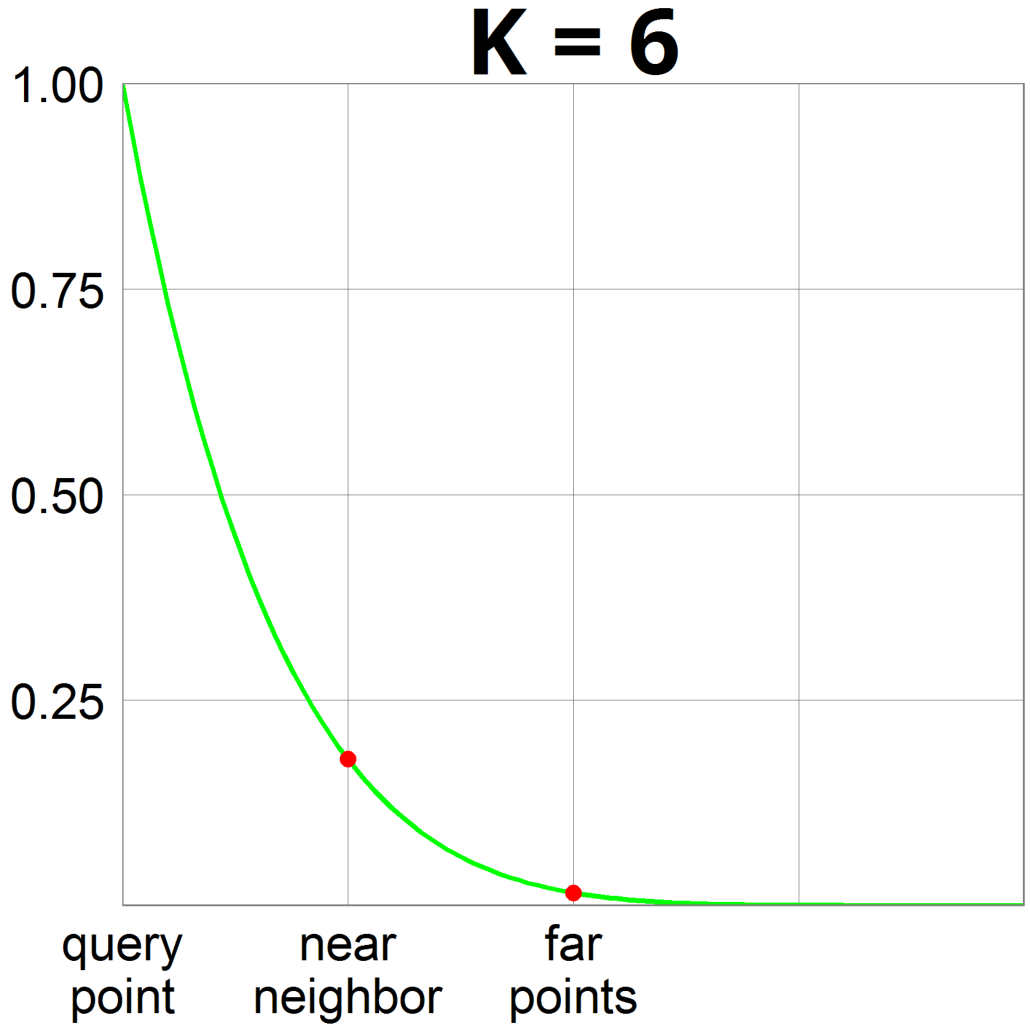
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)



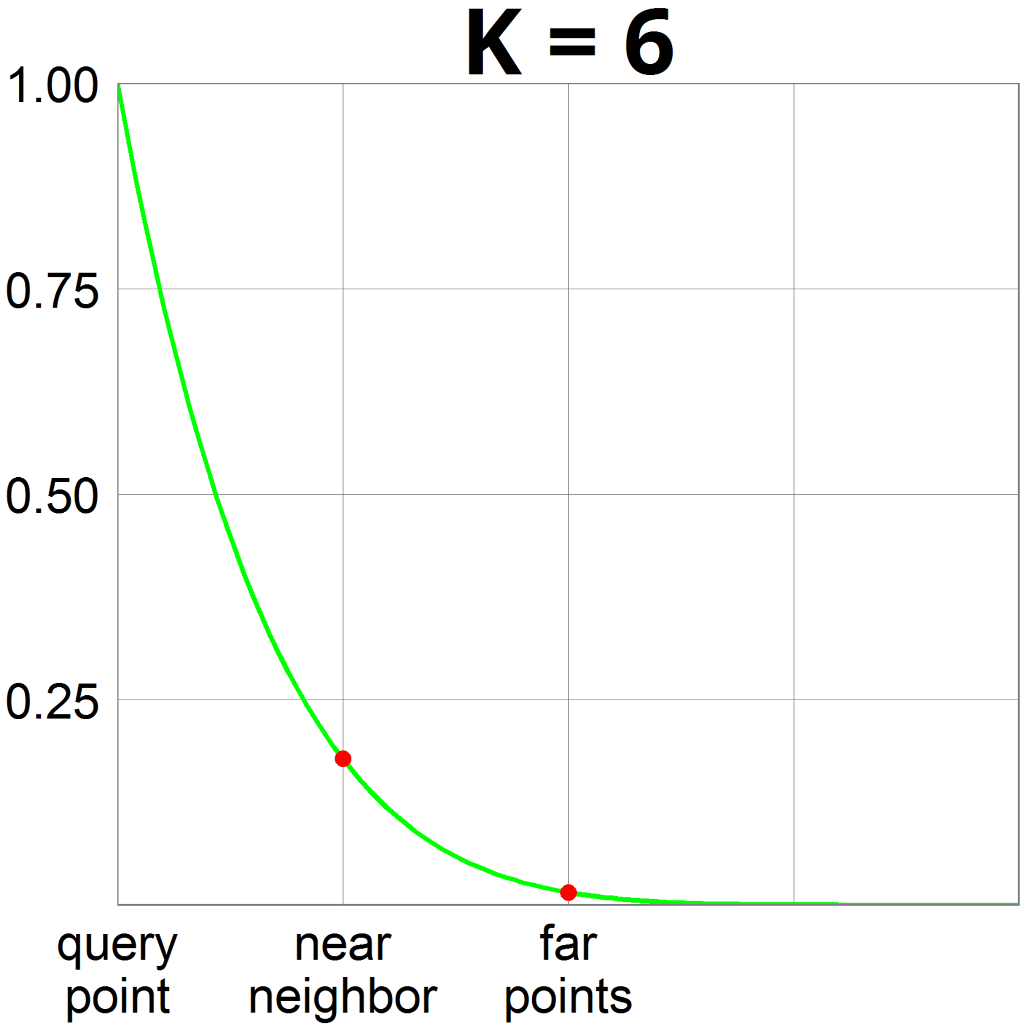
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)



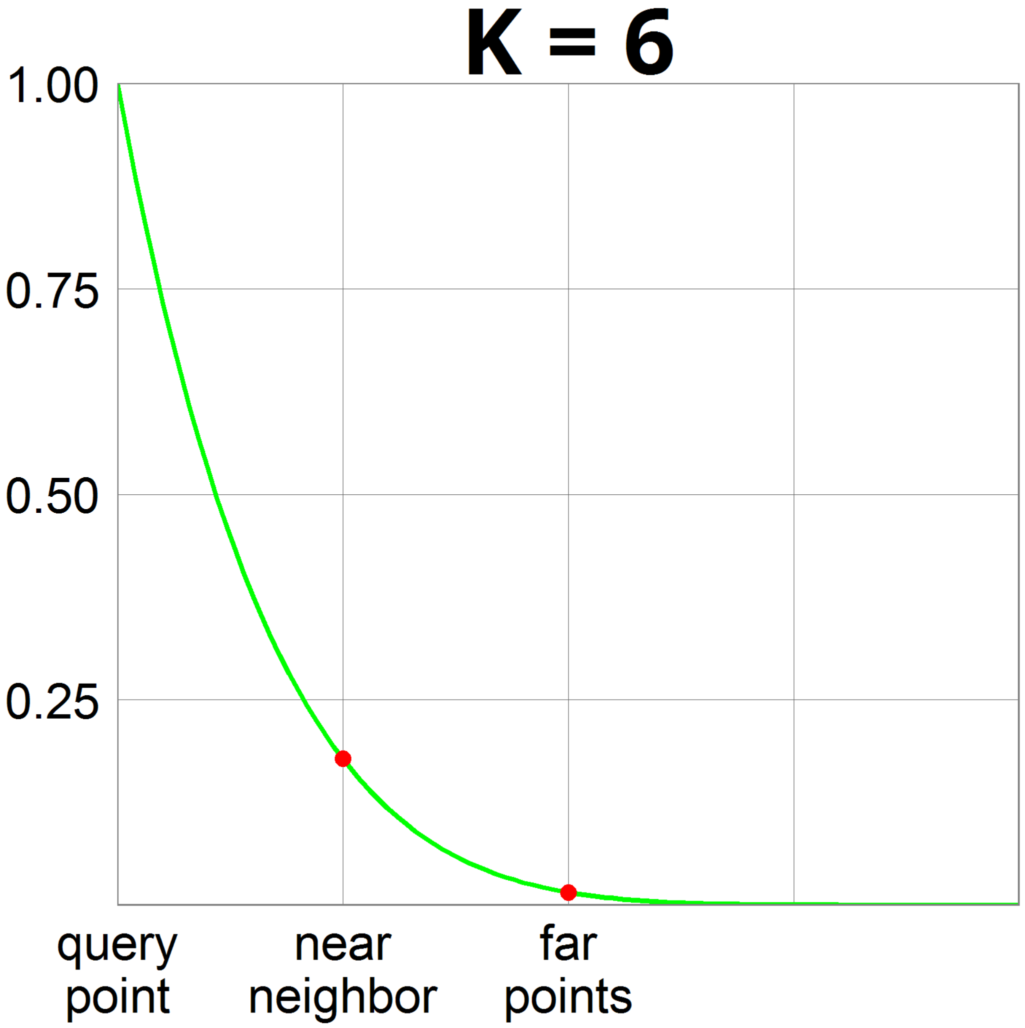
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)
- If **$0.5^K \sim 1/n$**, then query time is **O(1)**



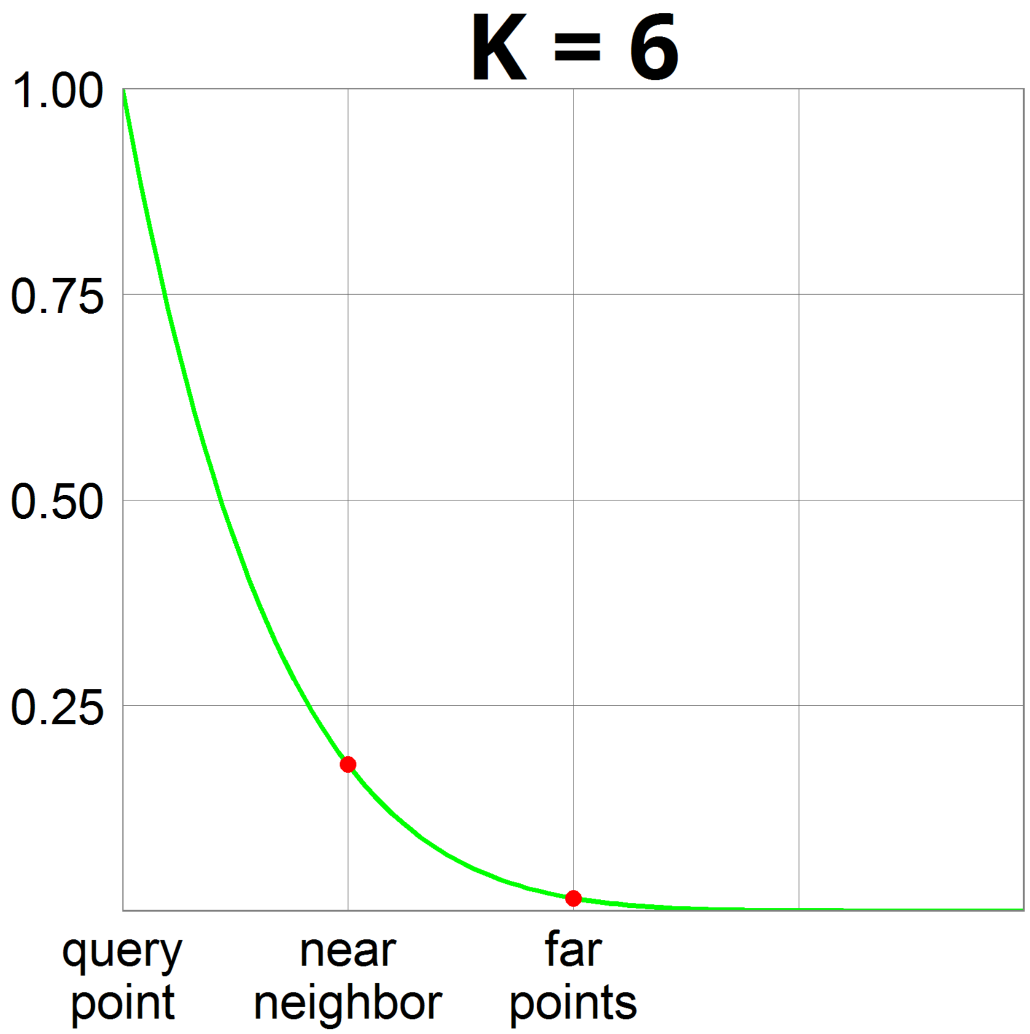
Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)
- If **0.5^K ~ 1/n**, then query time is **O(1)**
- Collides with near neighbor with probability **0.75^K ~ $1/n^{0.42}$**
- Thus, need **L = O($n^{0.42}$)** tables to boost the success probability to **0.99**



Using LSH to solve ANN

- **K** hash functions at once (**p** into **(h_1(p), …, h_K(p))**)
- If **$0.5^K \sim 1/n$**, then query time is **O(1)**
- Collides with near neighbor with probability **$0.75^K \sim 1/n^{0.42}$**
- Thus, need **L = $O(n^{0.42})$** tables to boost the success probability to **0.99**
- Overall: **$O(n^{1.42})$** space, **$O(n^{0.42})$** query time, **K·L** hyperplanes



Using LSH to solve ANN (in general)

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space **$O(n^{1+\rho})$** and query time **$O(n^{\rho})$**, where

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space **$O(n^{1+\rho})$** and query time **$O(n^{\rho})$**, where

$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space $\mathbf{O(n^{1+\rho})}$ and query time $\mathbf{O(n^{\rho})}$, where

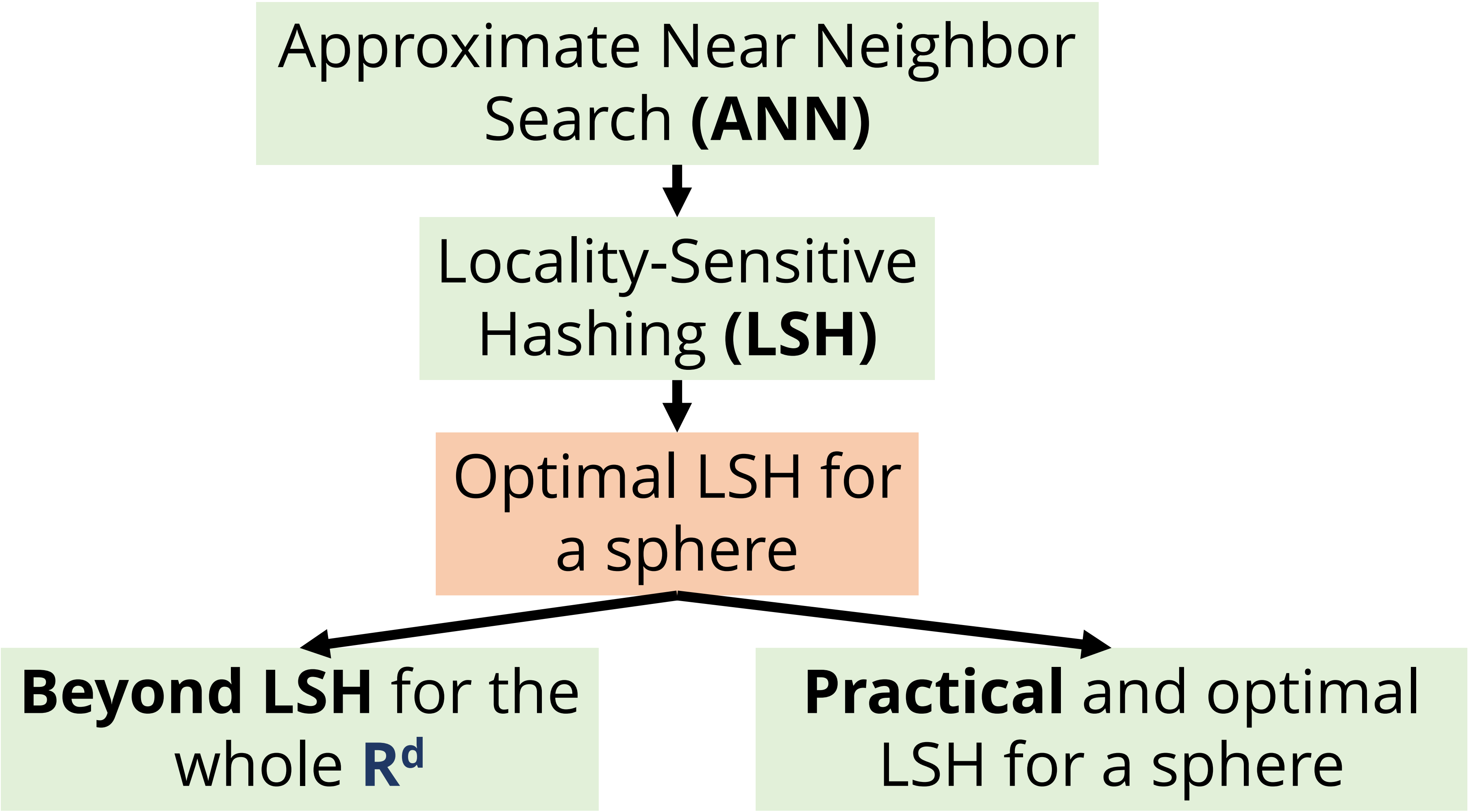
$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

Recap:

- $\mathbf{p_1}$ is collision probability for close pairs
- $\mathbf{p_2}$ — for far pairs

Outline

Outline


Approximate Near Neighbor Search **(ANN)**
Locality-Sensitive Hashing **(LSH)**
Optimal LSH for a sphere
Beyond LSH for the whole **R^d**
Practical and optimal LSH for a sphere

Better than Hyperplane LSH?

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**
  - **[Andoni, R ??]:** this is tight for the hashing-based approaches!

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**
  - **[Andoni, R ??]:** this is tight for the hashing-based approaches!
  - Works for the general case of ANN on a sphere!

Optimal LSH family: Voronoi LSH

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$g_1, g_2, \ldots, g_T$$

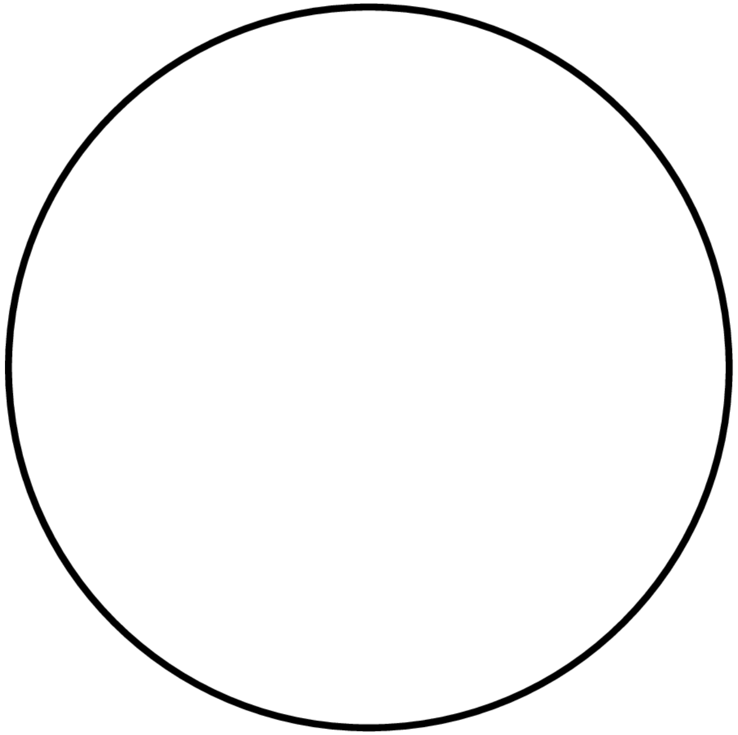
- Hash **p** into $h(p) = \mathrm{argmax}_{1 \le i \le T} \langle p, g_i \rangle$

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

- Hash **p** into $\mathbf{h(p) = \operatorname{argmax}_{1 \leq i \leq T} \langle p, g_i \rangle}$

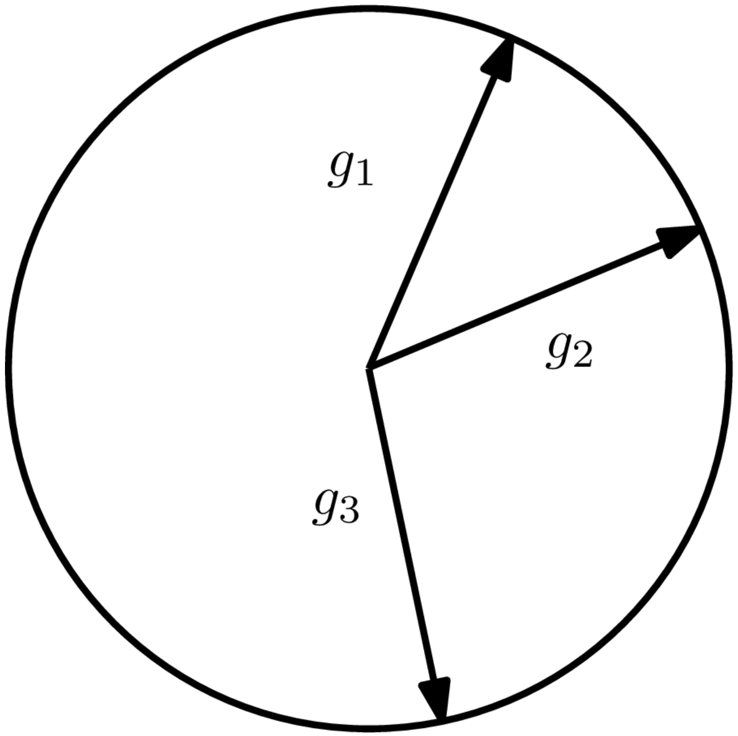


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$g_1, g_2, \ldots, g_T$$

- Hash **p** into $h(p) = \mathrm{argmax}_{1 \le i \le T} \langle p, g_i \rangle$

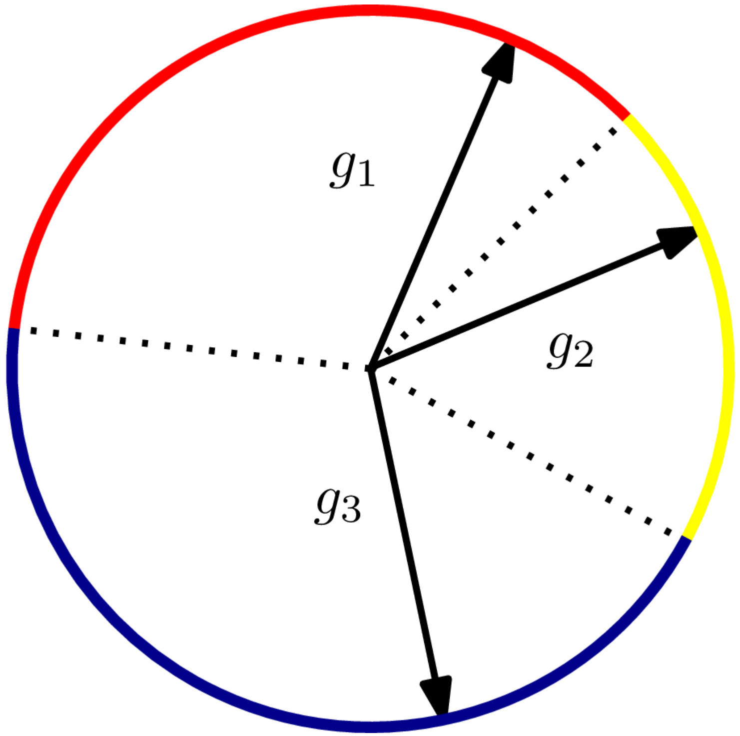


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$g_1, g_2, \ldots, g_T$$

- Hash **p** into $h(p) = \text{argmax}_{1 \le i \le T} \langle p, g_i \rangle$

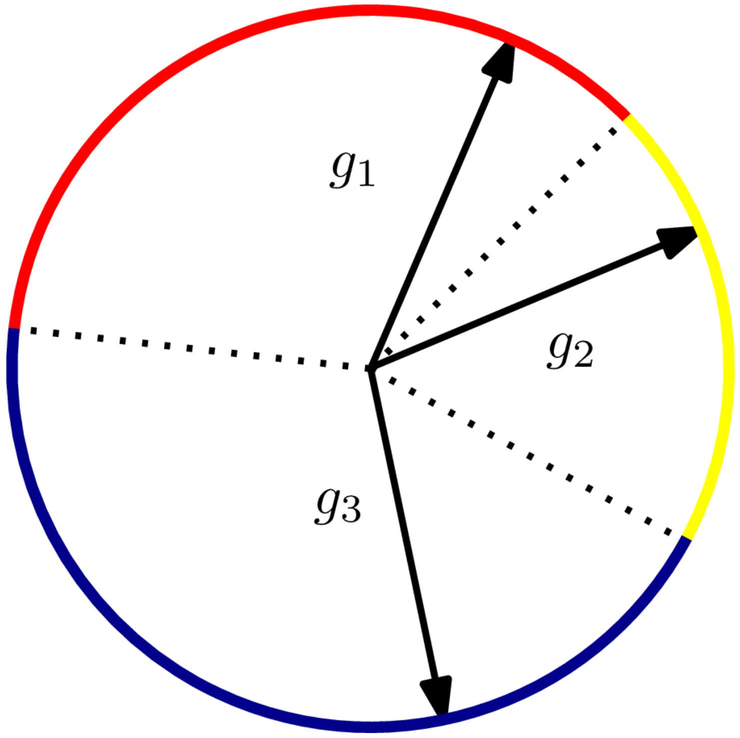


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

- Hash **p** into $\mathbf{h(p) = argmax_{1 \le i \le T} \langle p, g_i \rangle}$
- $\mathbf{T = 2}$ is simply Hyperplane LSH



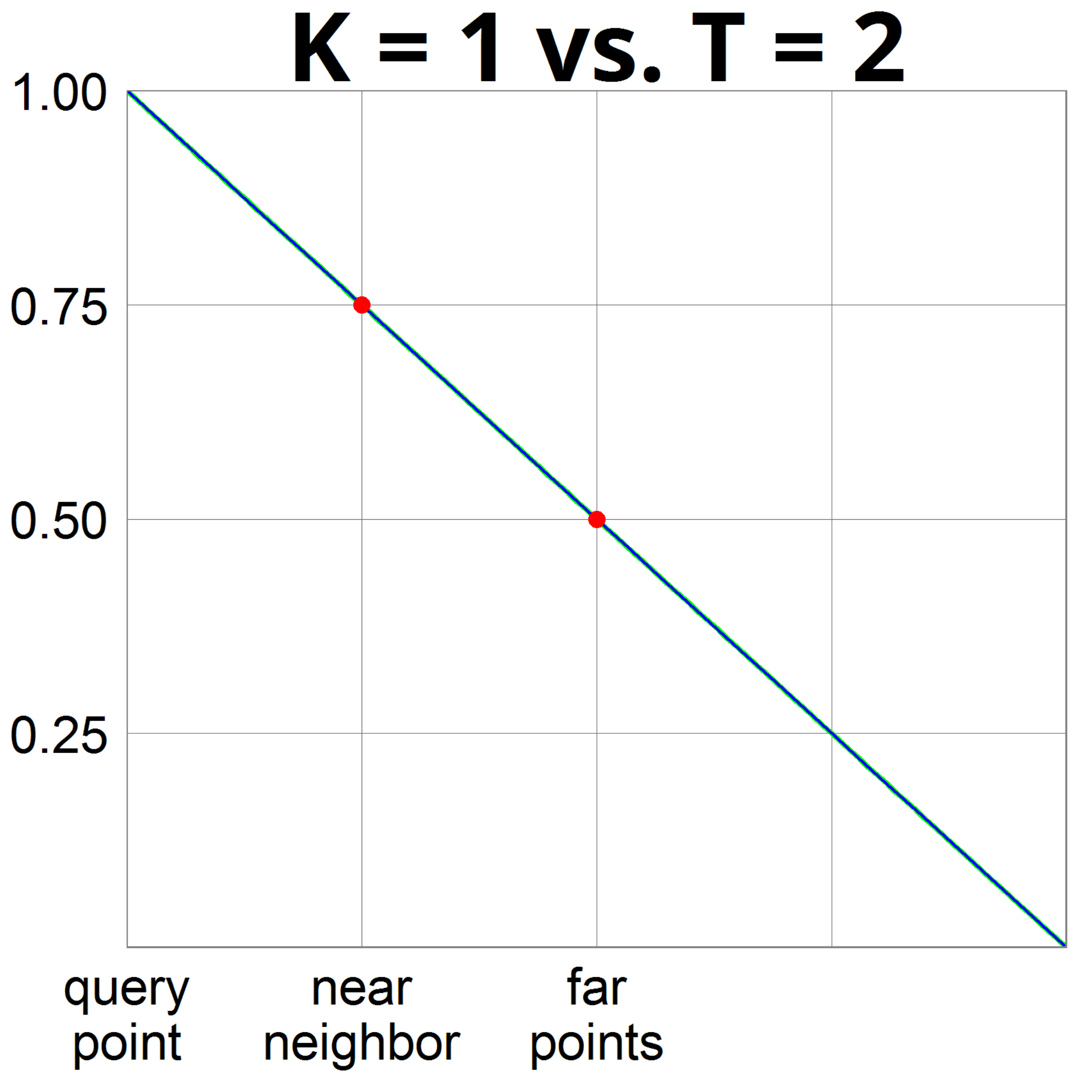
Hyperplane LSH vs. Voronoi LSH

Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **$T = 2^K$** (in both cases **K**-bit hashes)

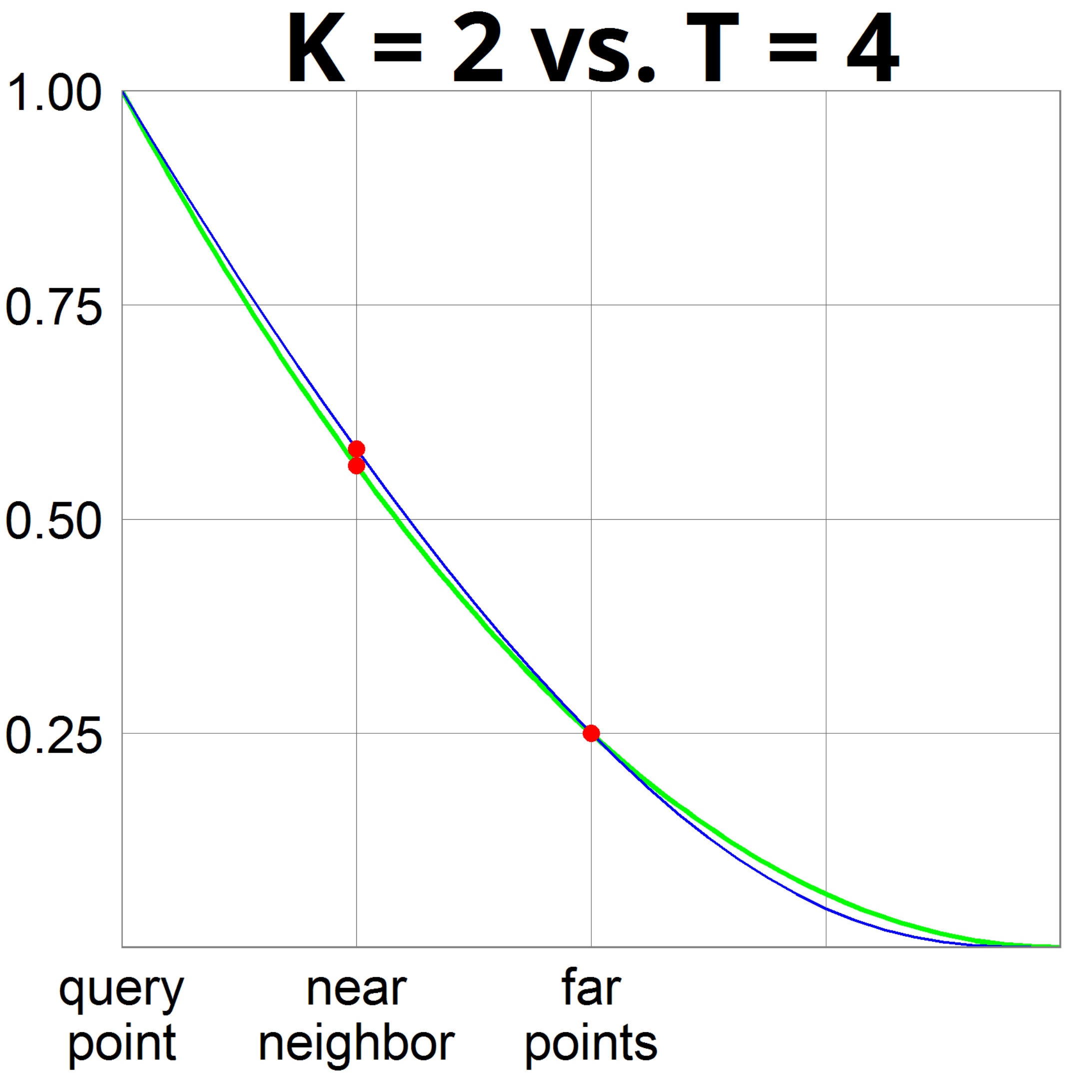
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **$T = 2^K$** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

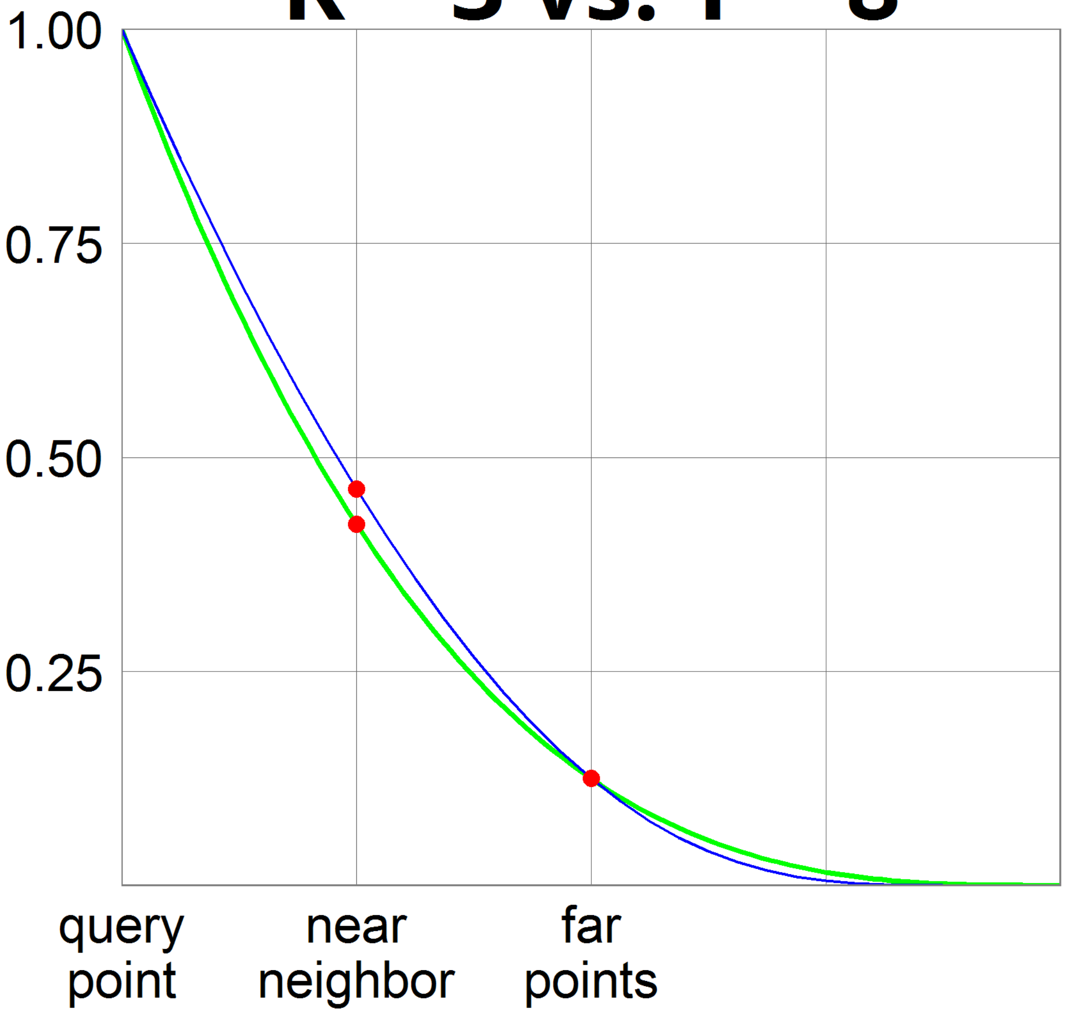
- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

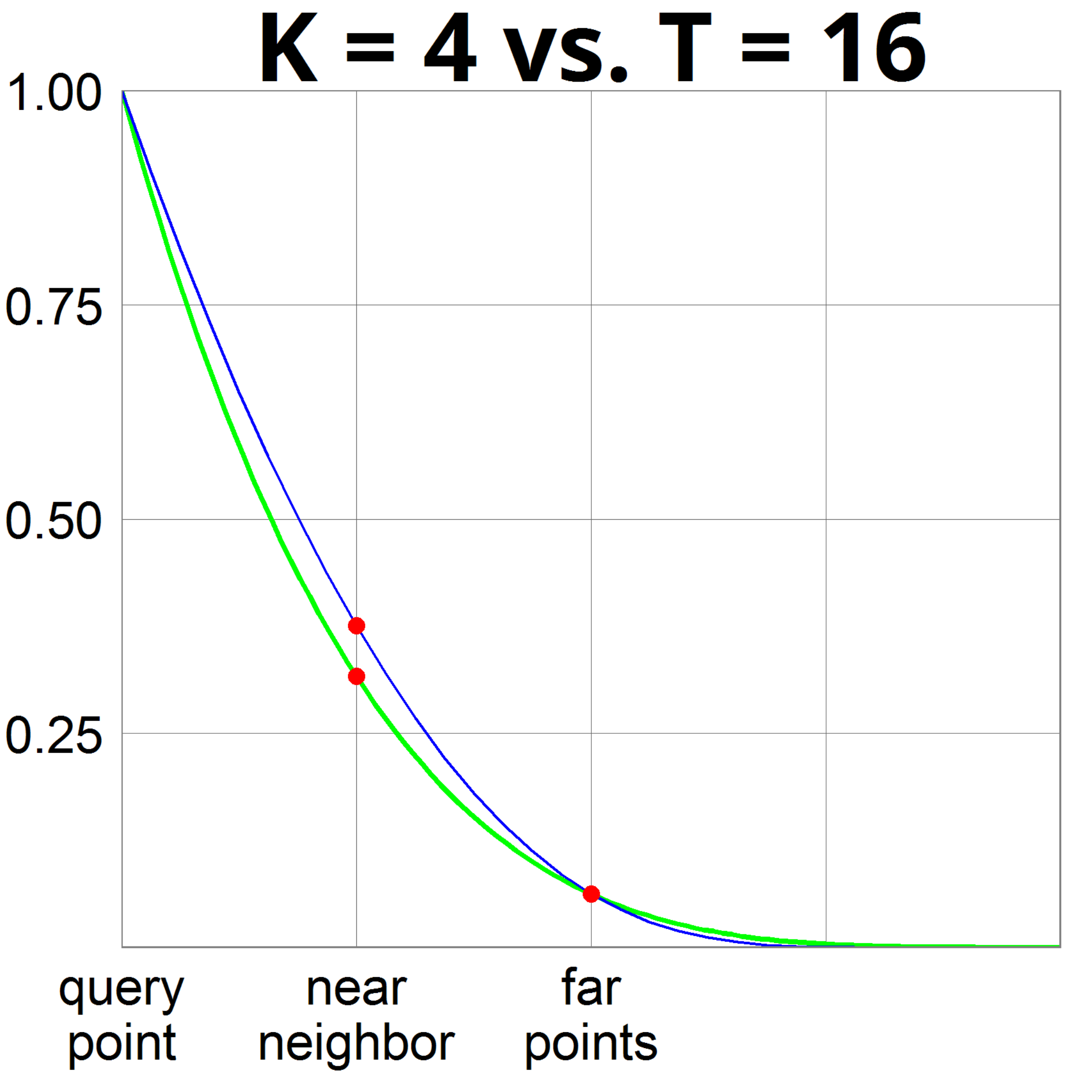
- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)

K = 3 vs. T = 8



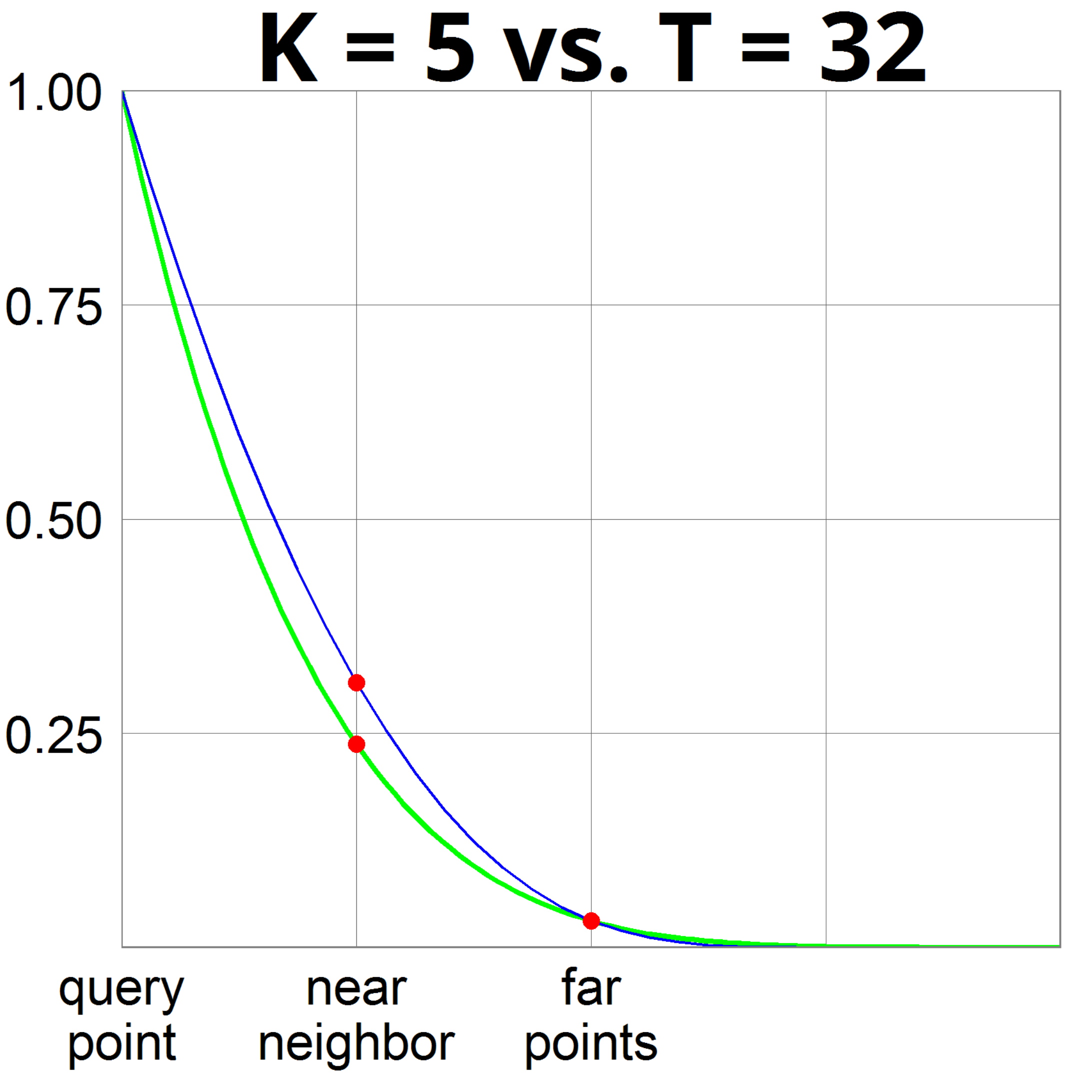
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^{K}** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

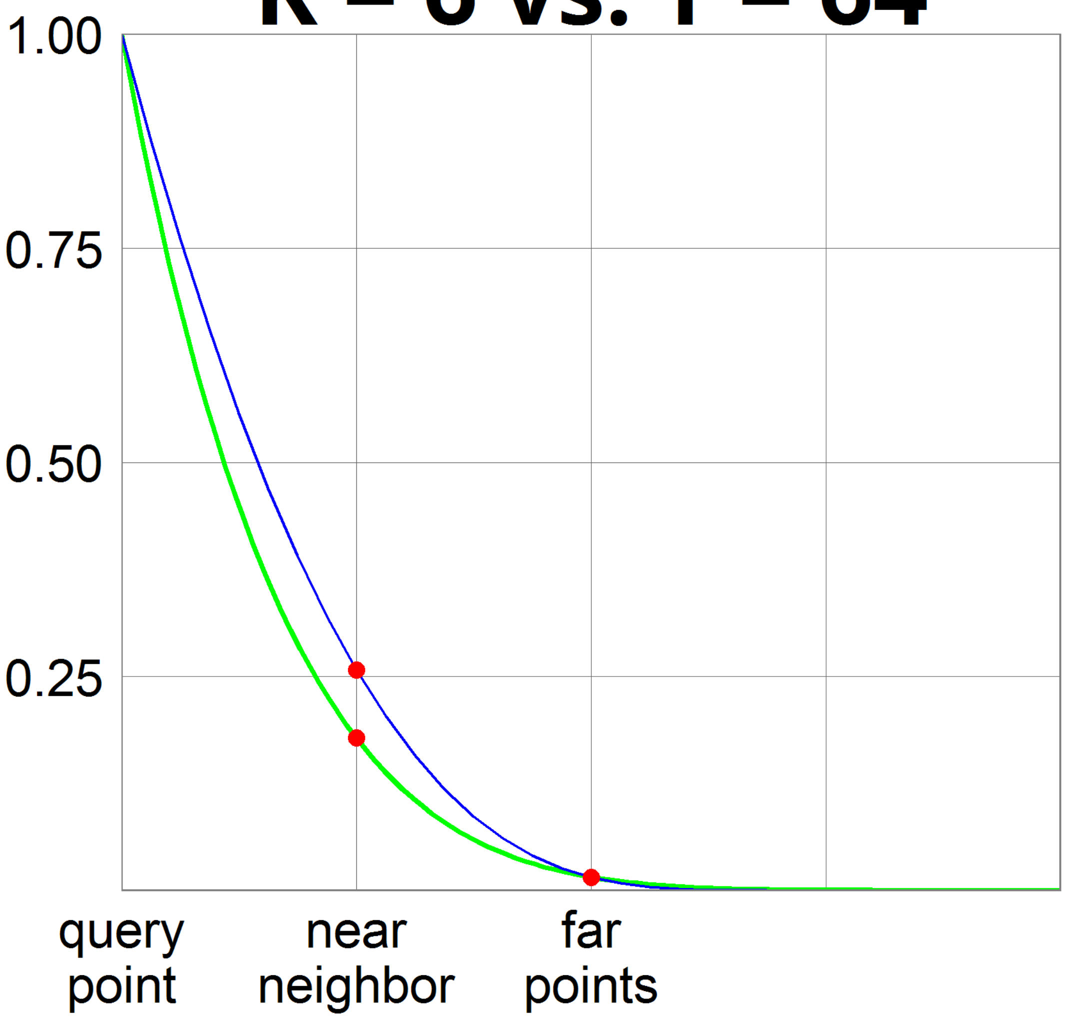
- Let us compare **K** hyperplanes vs. Voronoi LSH with **$T = 2^K$** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)

K = 6 vs. T = 64



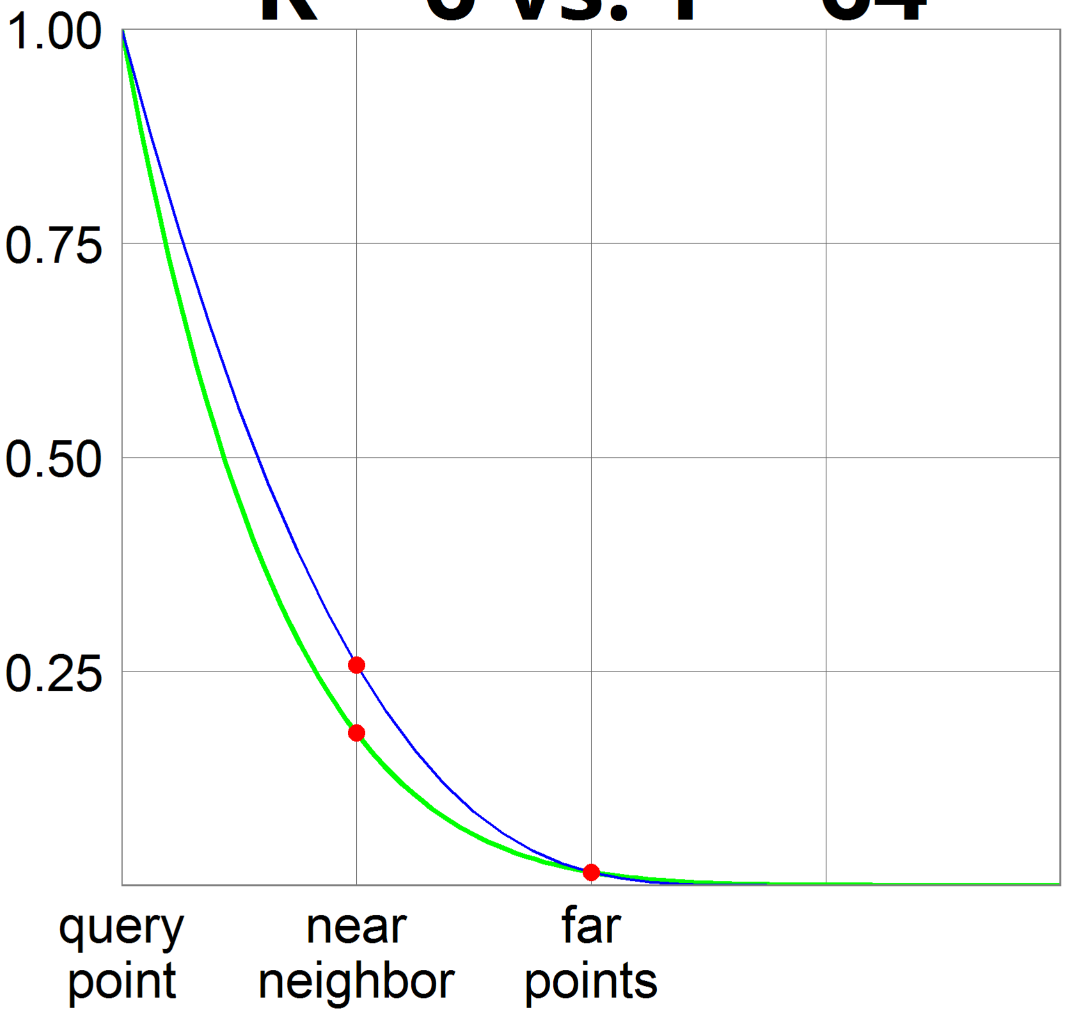
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **$T = 2^K$** (in both cases **K**-bit hashes)
- As **T** grows, the gap between Hyperplane LSH and Voronoi LSH increases and

$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

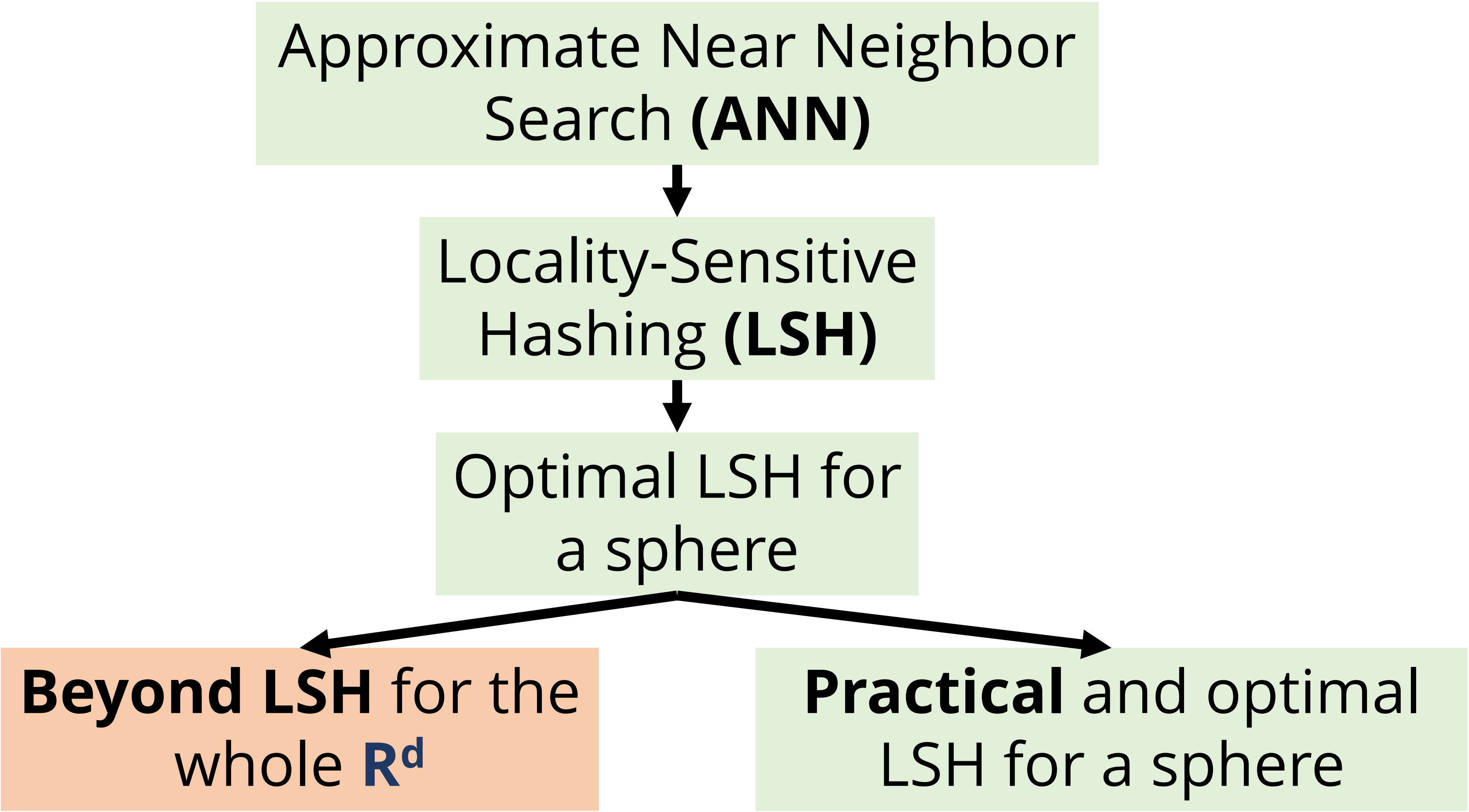
approaches **0.18**

K = 6 vs. T = 64

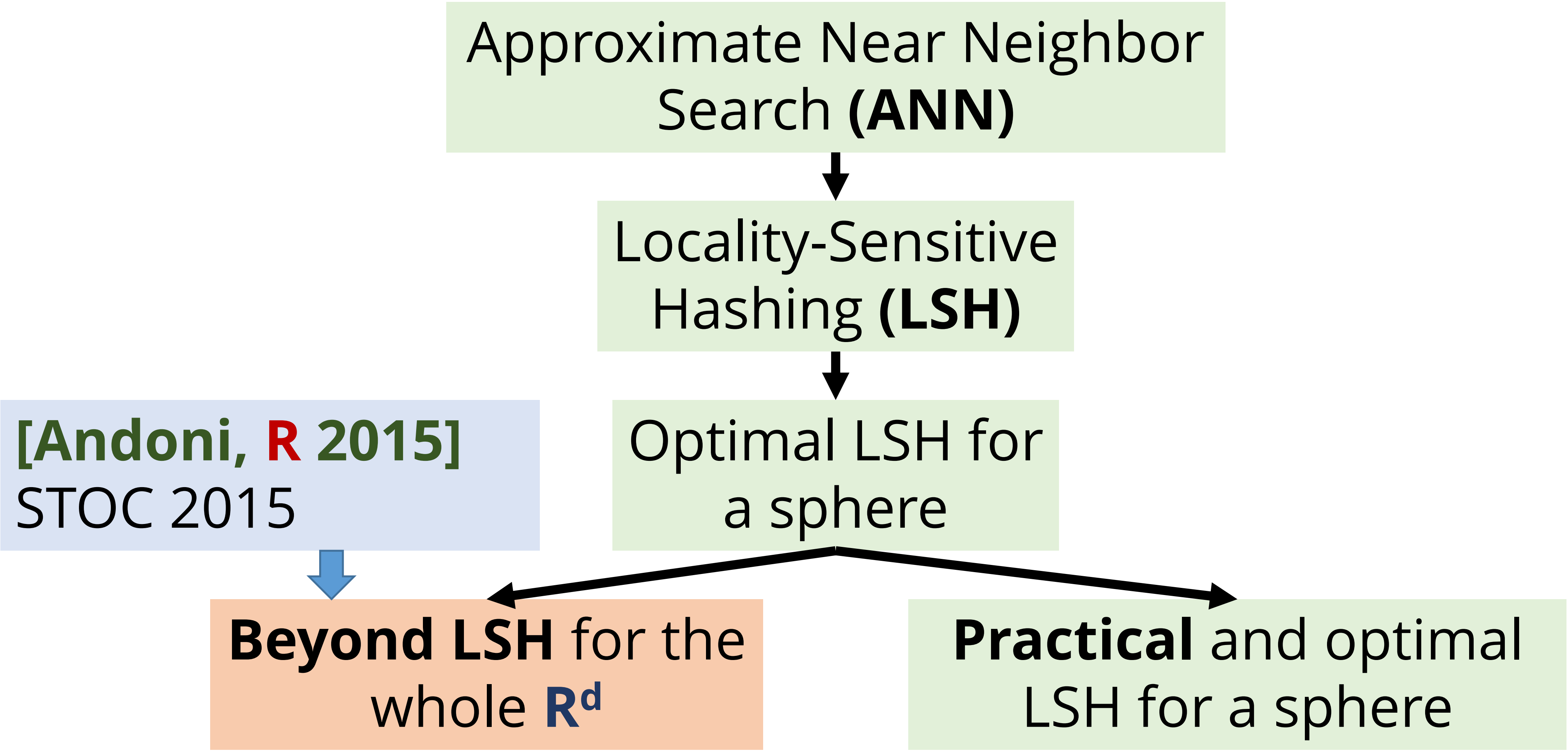


Outline

Outline


Approximate Near Neighbor Search **(ANN)**
Locality-Sensitive Hashing **(LSH)**
Optimal LSH for a sphere
Beyond LSH for the whole **R^d**
Practical and optimal LSH for a sphere

Outline


Approximate Near Neighbor Search **(ANN)**
Locality-Sensitive Hashing **(LSH)**
Optimal LSH for a sphere
[Andoni, R 2015]
STOC 2015
Beyond LSH for the whole **R**^{d}
Practical and optimal LSH for a sphere

Bounds on LSH

Bounds on LSH

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\leq \mathbf{1/c^2 + o(1)}$	1/4	**[Andoni, Indyk 2006]**
	$\geq 1/c^2 - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
Manhattan, Hamming (ℓ_1)	$\leq \mathbf{1/c}$	1/2	**[Indyk, Motwani 1998]**
	$\geq 1/c - o(1)$		**[O'Donnell, Wu, Zhou 2011]**

Bounds on LSH

Space $O(n^{3/2})$, query time $O(n^{1/2})$

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\le \mathbf{1/c^2 + o(1)}$	1/4	**[Andoni, Indyk 2006]**
	$\ge 1/c^2 - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
Manhattan, Hamming (ℓ_1)	$\le \mathbf{1/c}$	1/2	**[Indyk, Motwani 1998]**
	$\ge 1/c - o(1)$		**[O'Donnell, Wu, Zhou 2011]**

Bounds on LSH

Space $O(n^{3/2})$, query time $O(n^{1/2})$

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\leq \mathbf{1/c^2 + o(1)}$	1/4	[Andoni, Indyk 2006]
	$\geq 1/c^2 - o(1)$		[O'Donnell, Wu, Zhou 2011]
Manhattan, Hamming (ℓ_1)	$\leq \mathbf{1/c}$	1/2	[Indyk, Motwani 1998]
	$\geq 1/c - o(1)$		[O'Donnell, Wu, Zhou 2011]

Can one improve upon LSH?

Bounds on LSH

Space $O(n^{3/2})$, query time $O(n^{1/2})$

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\leq \mathbf{1/c^2 + o(1)}$	1/4	**[Andoni, Indyk 2006]**
	$\geq 1/c^2 - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
Manhattan, Hamming (ℓ_1)	$\leq \mathbf{1/c}$	1/2	**[Indyk, Motwani 1998]**
	$\geq 1/c - o(1)$		**[O'Donnell, Wu, Zhou 2011]**

Can one improve upon LSH?

Yes!

How to do better than LSH?

How to do better than LSH?

- **Main idea: data-dependent space partitions**

How to do better than LSH?

- **Main idea: data-dependent space partitions**
- A distribution over partitions **R** is **(r, cr, p_1, p_2)-sensitive** if **for every p, q**:
    - If $\|p - q\| \leq r$, then $\Pr_R[R(p) = R(q)] \geq p_1$
    - If $\|p - q\| \geq cr$, then $\Pr_R[R(p) = R(q)] \leq p_2$

How to do better than LSH?

- **Main idea: data-dependent space partitions**
- A distribution over partitions **R** is **(r, cr, p_1, p_2)-sensitive** if **for every p, q**:
  - If $\|p - q\| \leq r$, then $\Pr_R[R(p) = R(q)] \geq p_1$
  - If $\|p - q\| \geq cr$, then $\Pr_R[R(p) = R(q)] \leq p_2$
- **Too strong! Can assume that p is a data point!**
  - Exploit the geometry of **P** to design better partitions
  - Able to obtain **improvement for every P**

The result

The result

Optimal* data-dependent space partitions for the Euclidean and Manhattan/Hamming distances

* After proper formalization

The main result (quantitative)

The main result (quantitative)

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\leq 1/c^2 + o(1)$	1/4	**[Andoni, Indyk 2006]**
	$\geq 1/c^2 - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
Hamming (ℓ_1)	$\leq 1/c$	1/2	**[Indyk, Motwani 1998]**
	$\geq 1/c - o(1)$		**[O'Donnell, Wu, Zhou 2011]**

The main result (quantitative)

Distance metric	$\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$	$c = 2$	Reference
Euclidean (ℓ_2)	$\leq 1/c^2 + o(1)$	1/4	**[Andoni, Indyk 2006]**
	$\geq 1/c^2 - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
	$\mathbf{\frac{1}{2c^2 - 1} + o(1)}$	**1/7**	**[Andoni, R 2015]**
Hamming (ℓ_1)	$\leq 1/c$	1/2	**[Indyk, Motwani 1998]**
	$\geq 1/c - o(1)$		**[O'Donnell, Wu, Zhou 2011]**
	$\mathbf{\frac{1}{2c - 1} + o(1)}$	**1/3**	**[Andoni, R 2015]**

The plan

The plan

- Random datasets (**data-independent,** via Voronoi LSH)

The plan

- Random datasets (**data-independent,** via Voronoi LSH)
- **Worst-case dataset $\to$ randomly-looking parts (data-dependent)**

Random case

Random case

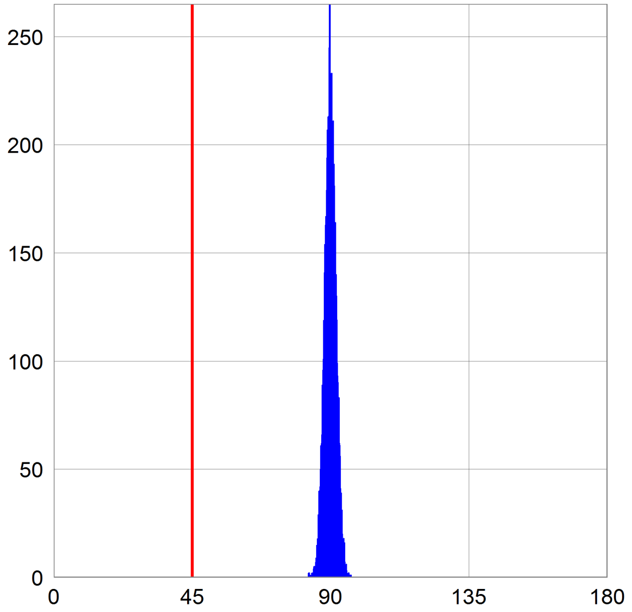
- W.l.o.g. points and queries lie on a sphere of radius **R**

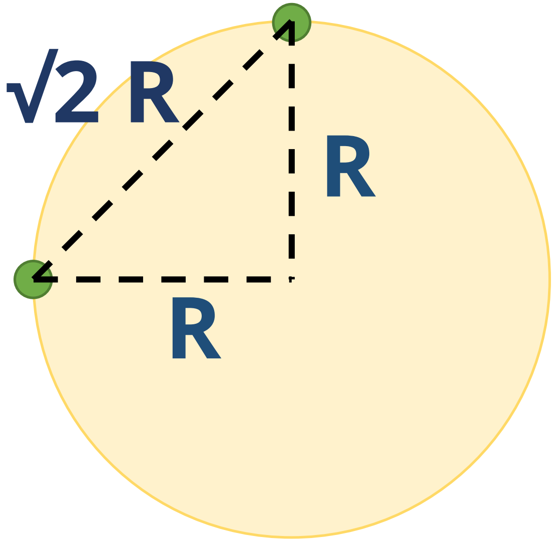
Random case

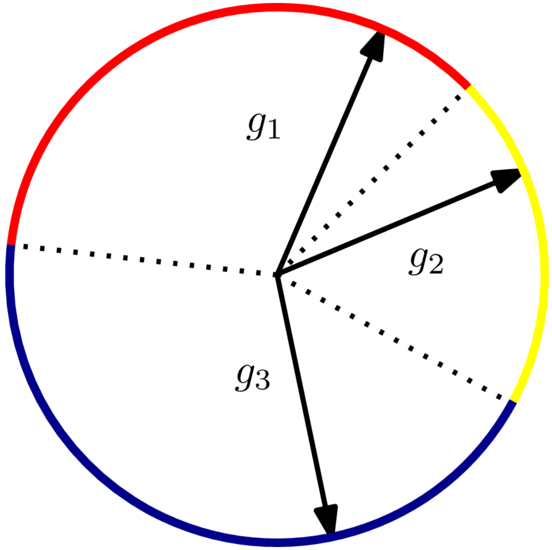
- W.l.o.g. points and queries lie on a sphere of radius **R**
- Random instance; near neighbors are planted within **$\sqrt{2}\, R/c$**

Random case

- W.l.o.g. points and queries lie on a sphere of radius **R**
- Random instance; near neighbors are planted within **√2 R/c**

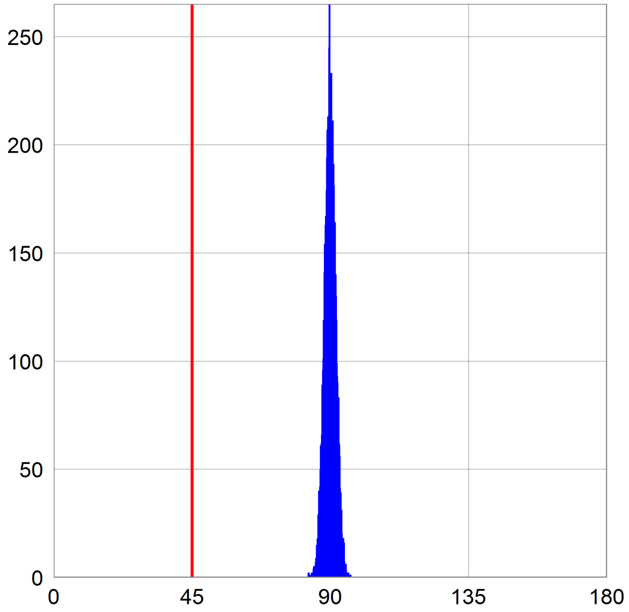


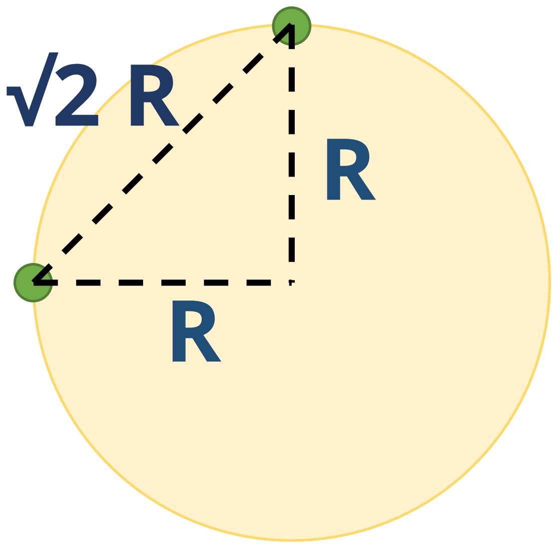


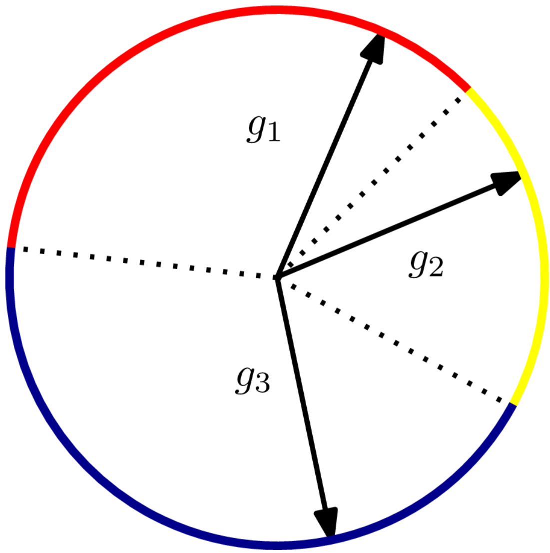


Random case

- W.l.o.g. points and queries lie on a sphere of radius **R**
- Random instance; near neighbors are planted within **√2 R/c**
- **Voronoi LSH** gives $\rho = \log(1/p_1) / \log(1/p_2) = 1 / (2c^2 - 1)$

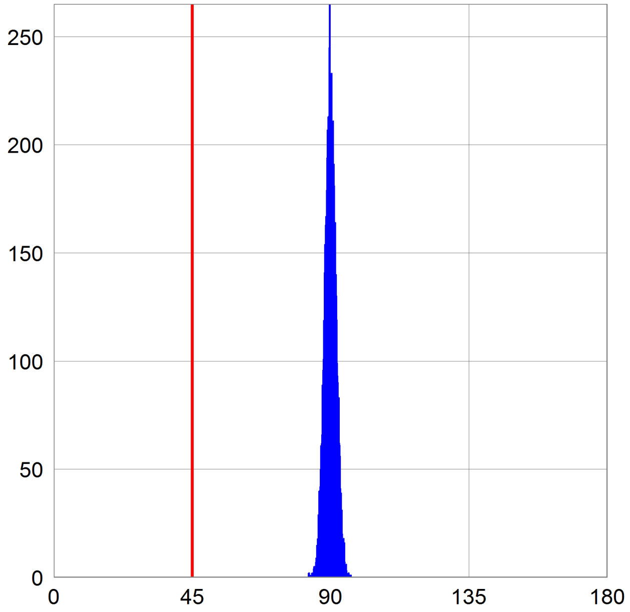


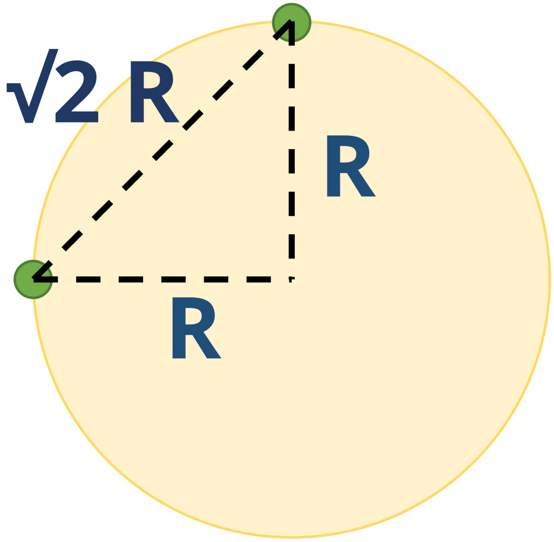


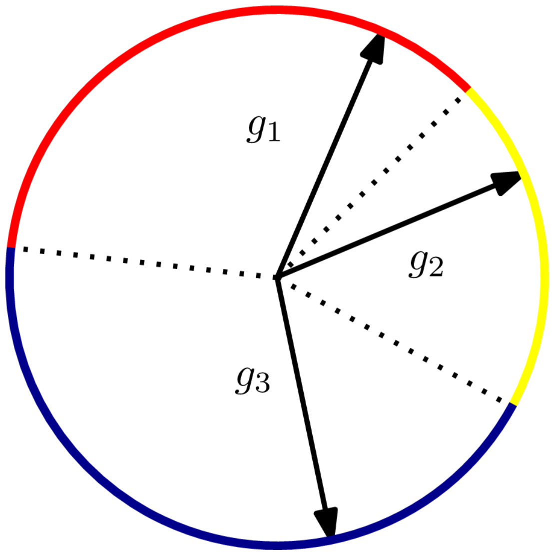


Random case

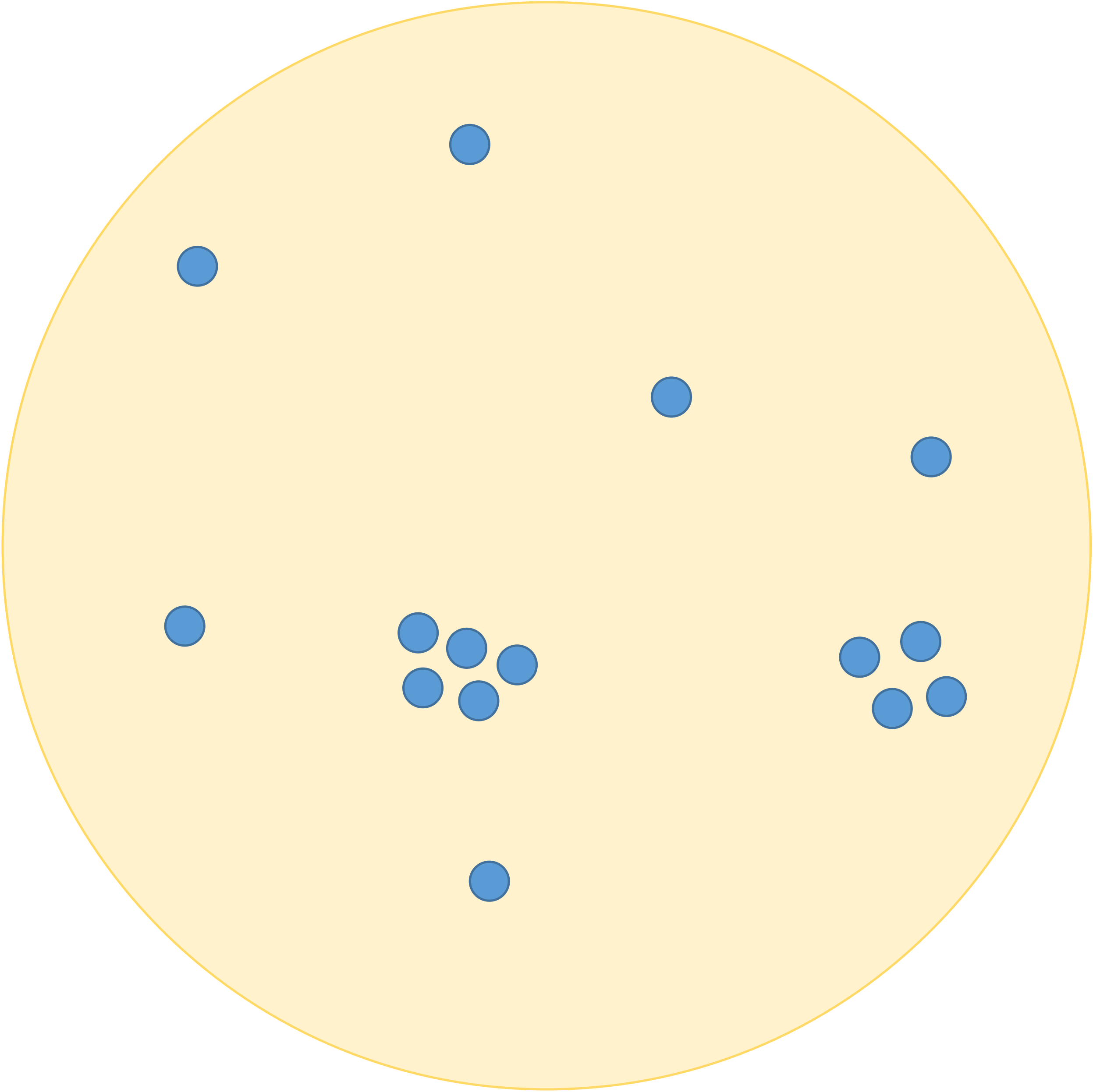
- W.l.o.g. points and queries lie on a sphere of radius **R**
- Random instance; near neighbors are planted within **√2 R/c**
- **Voronoi LSH** gives $\rho = \log(1/p_1) / \log(1/p_2) = 1 / (2c^2 - 1)$
- **What if the dataset does not look random?**
  - Voronoi LSH is suboptimal





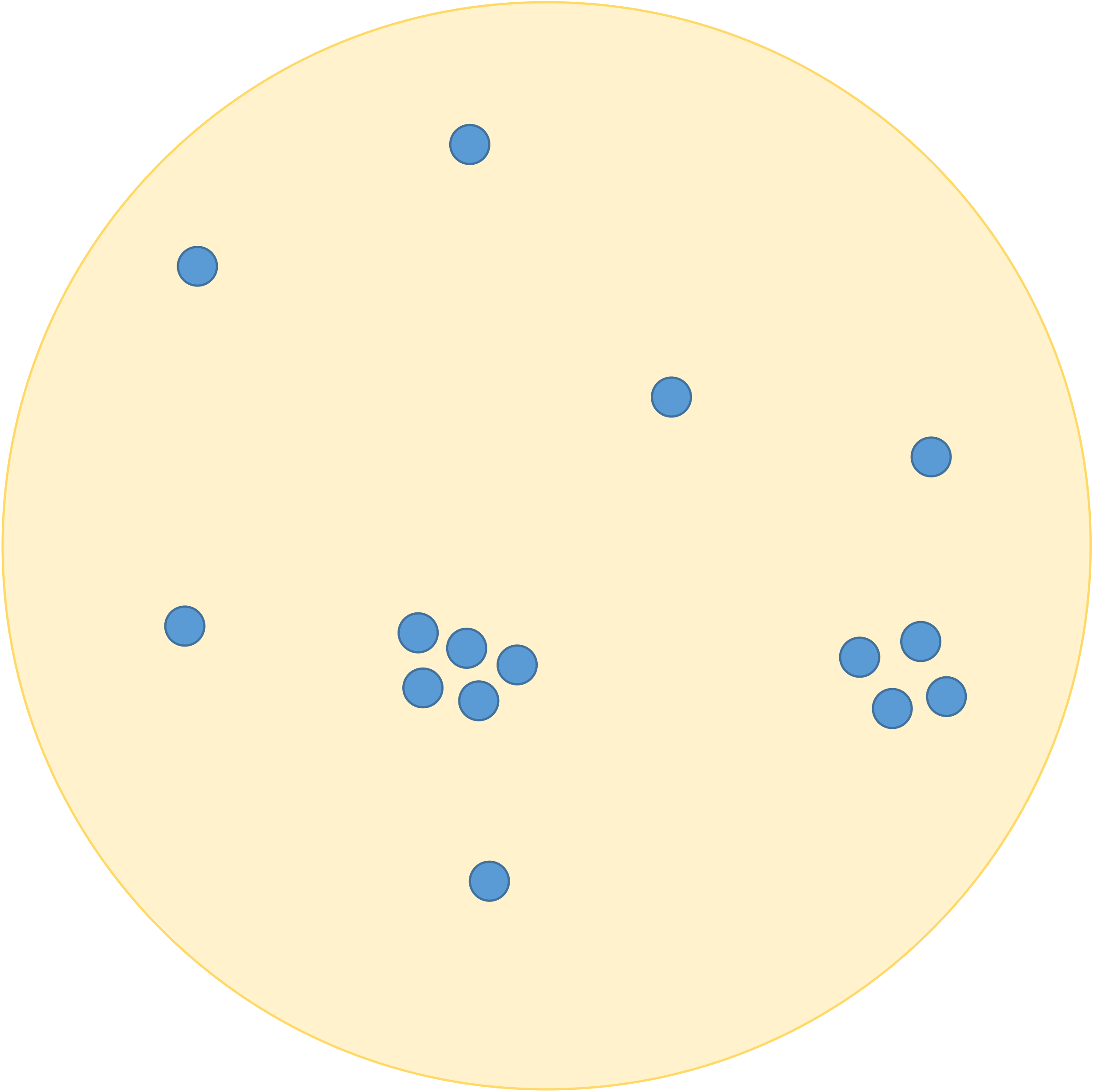


General case



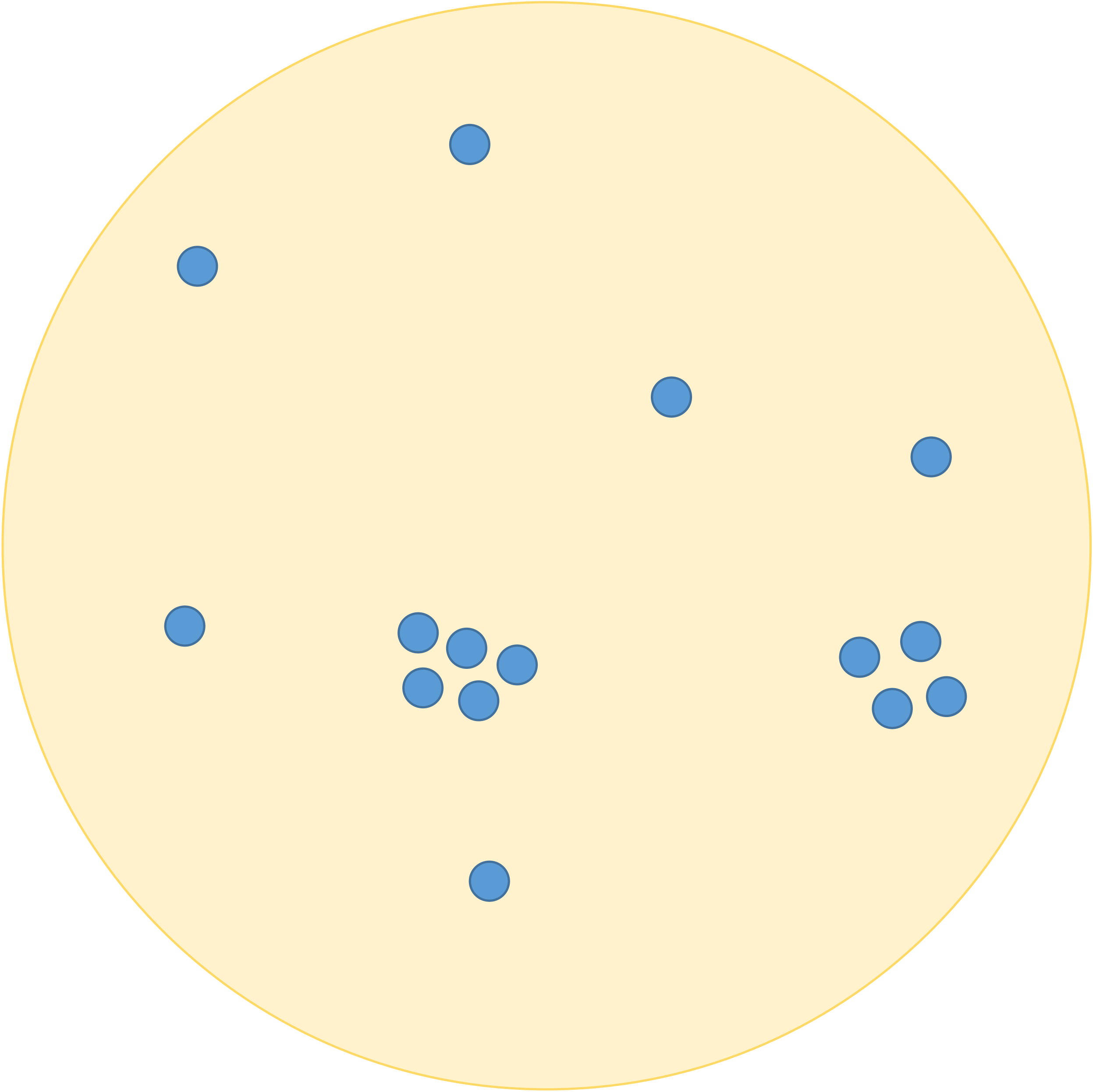
General case

- The dataset does not look random



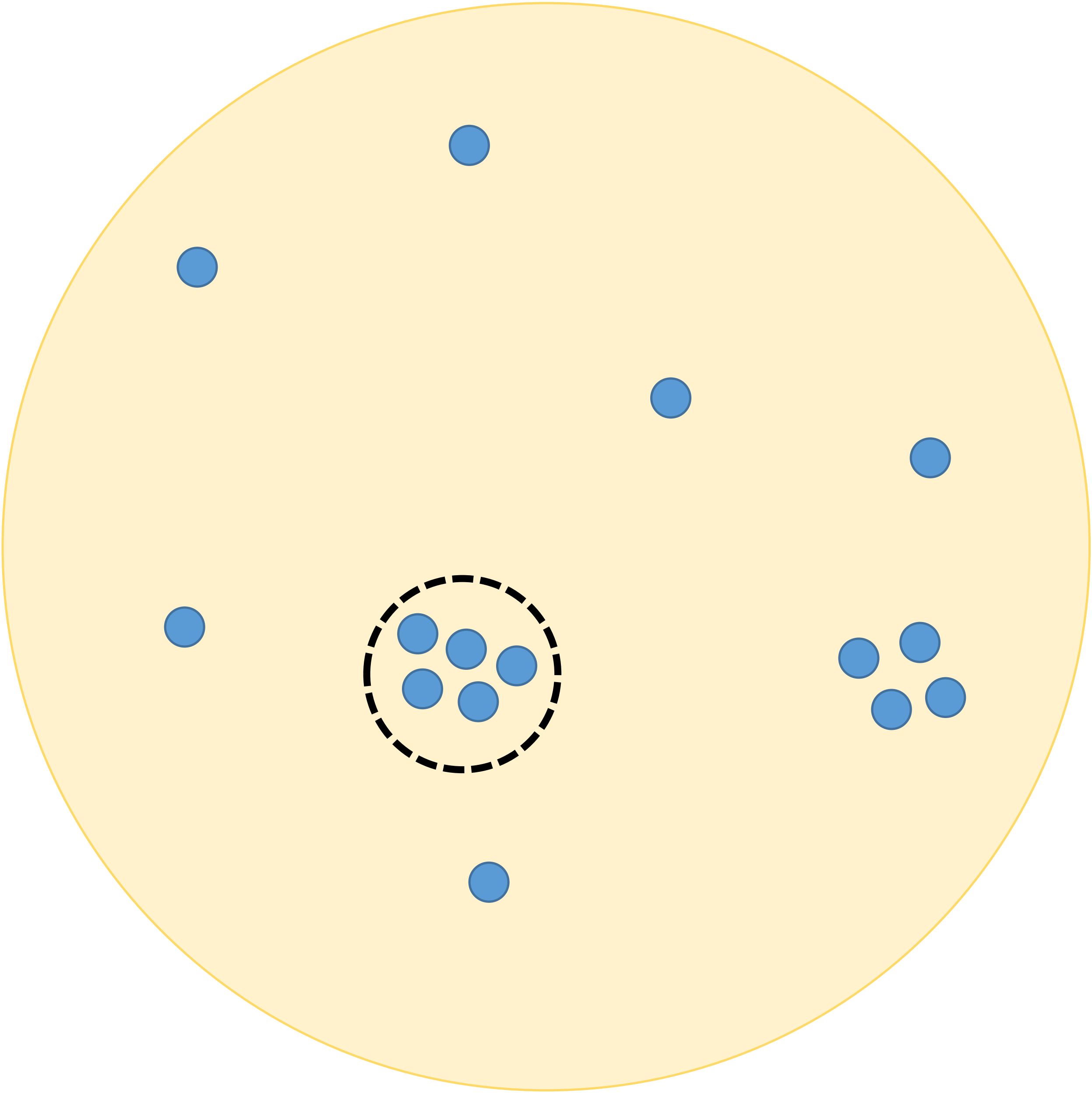
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1-\delta}}$ points—until there are none
  - Will handle them separately



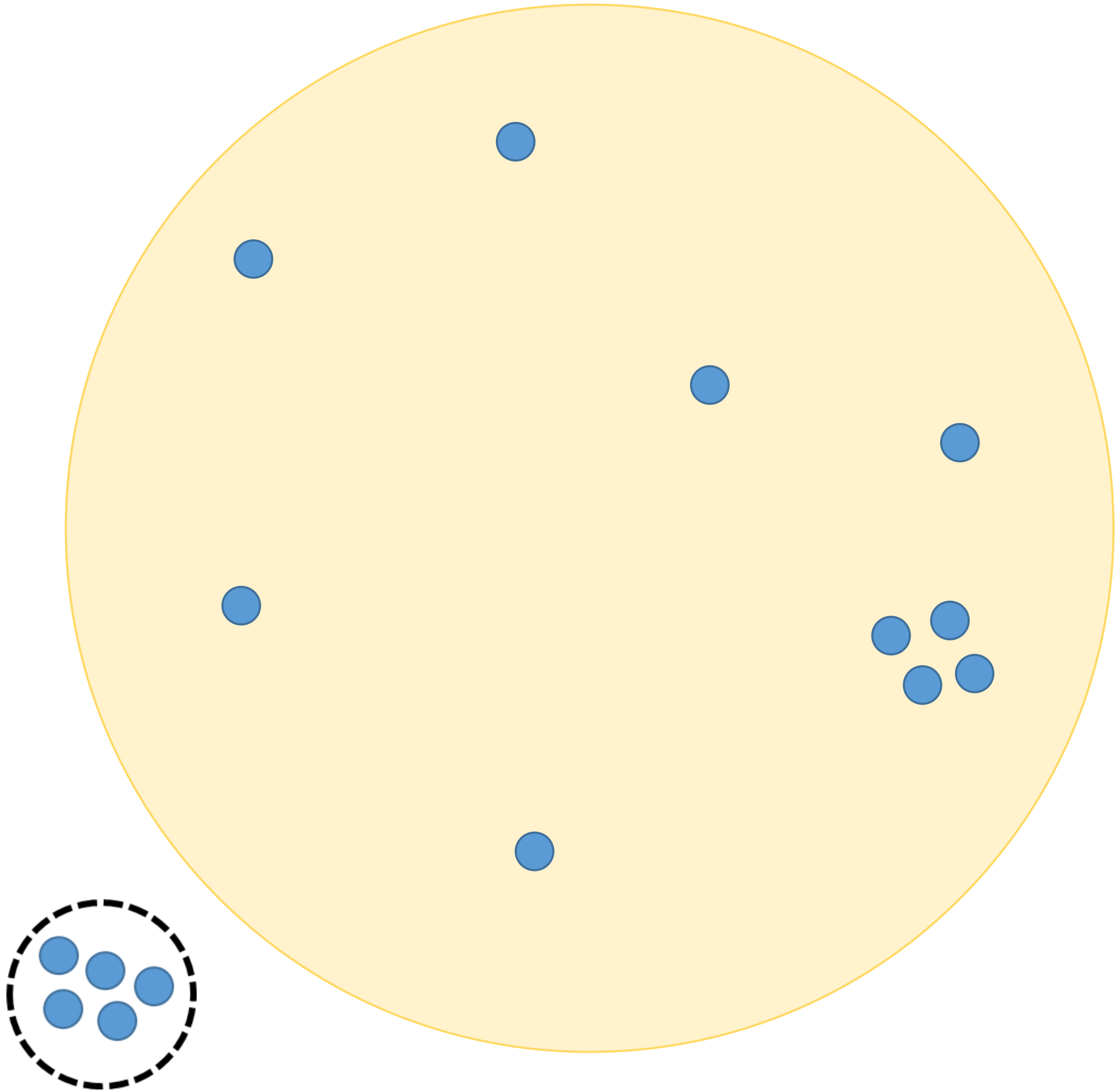
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1\text{-}\delta}}$ points—until there are none
  - Will handle them separately



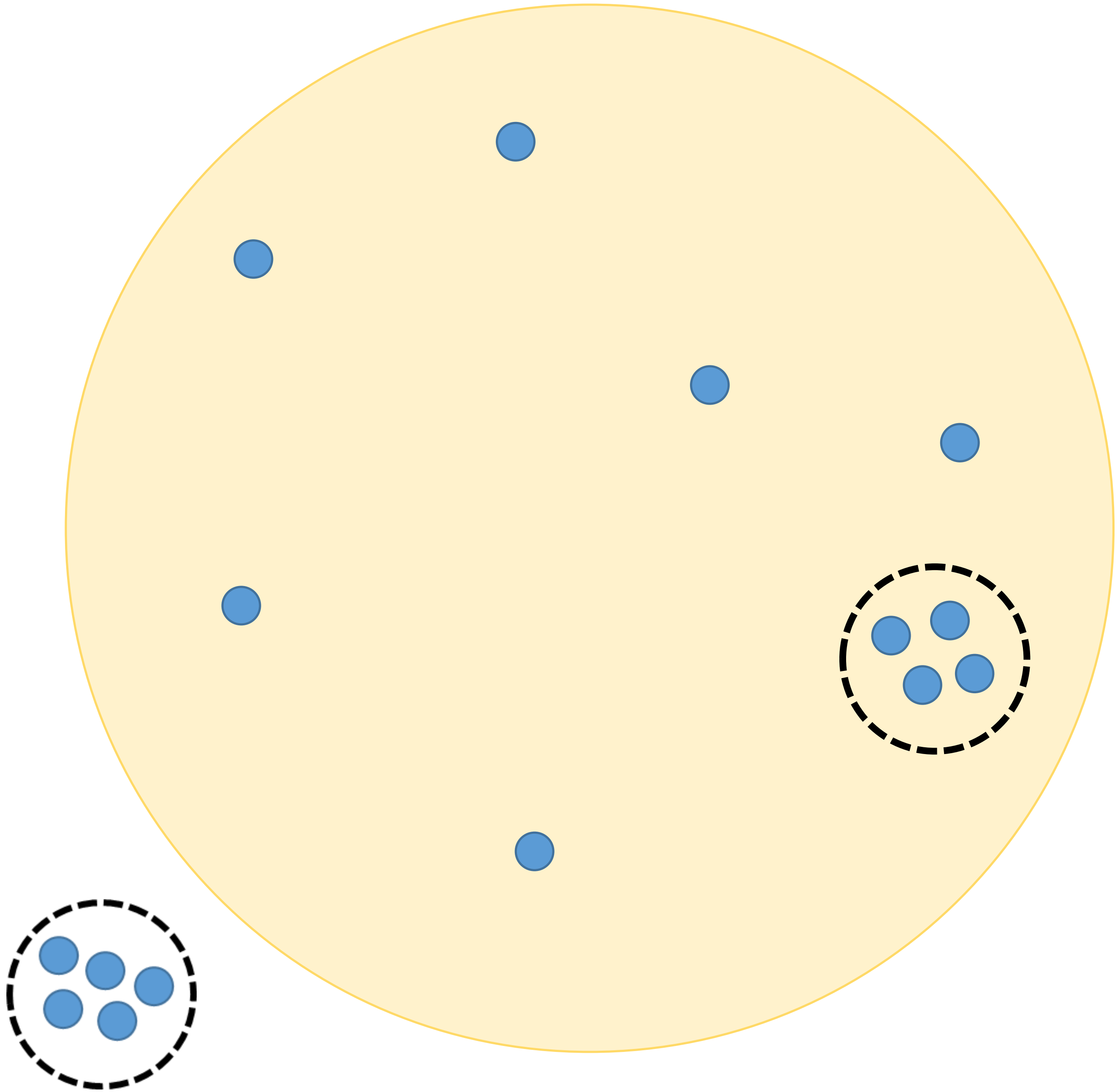
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1-\delta}}$ points—until there are none
  - Will handle them separately



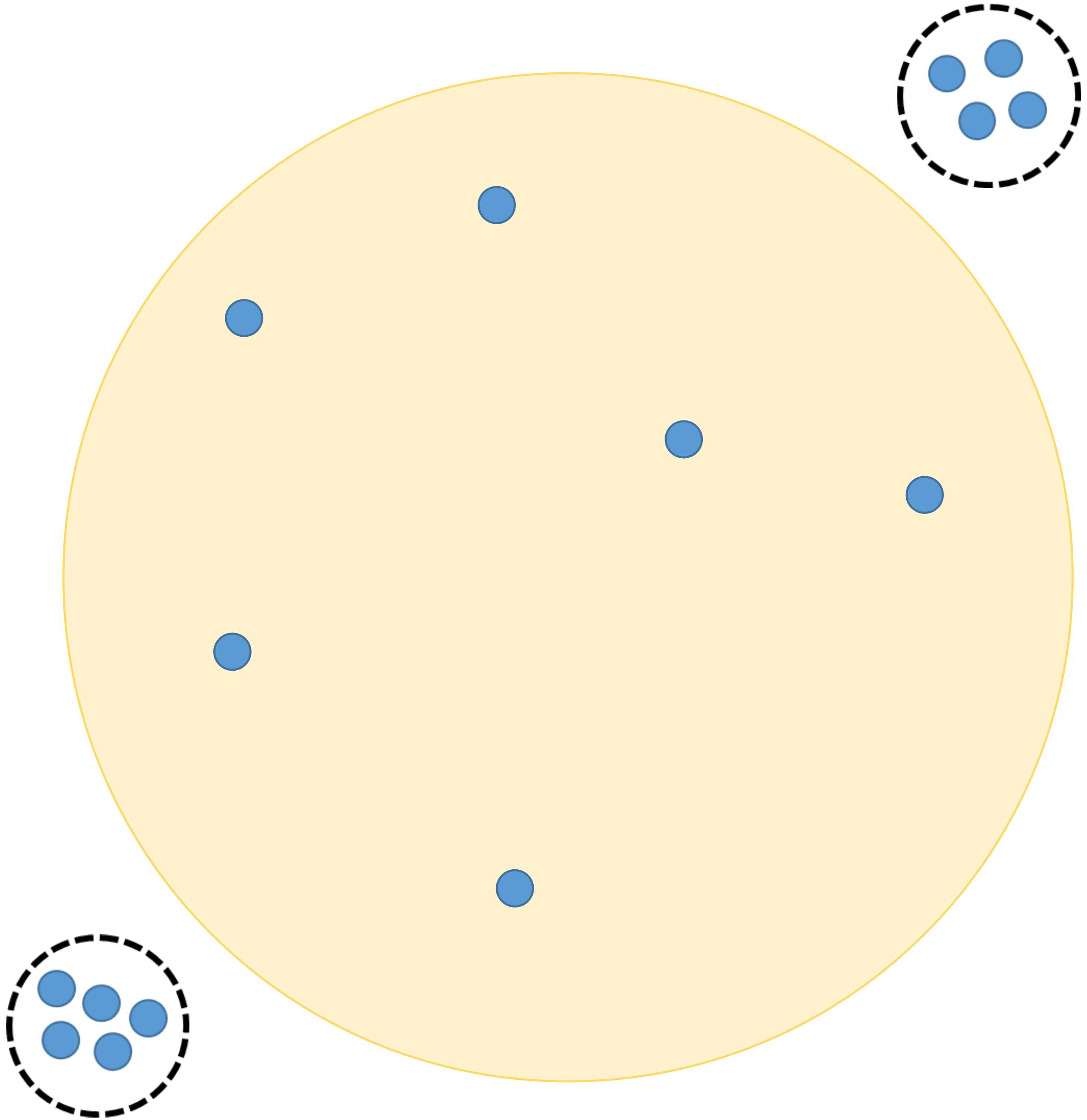
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with **$n^{1-\delta}$** points—until there are none
  - Will handle them separately



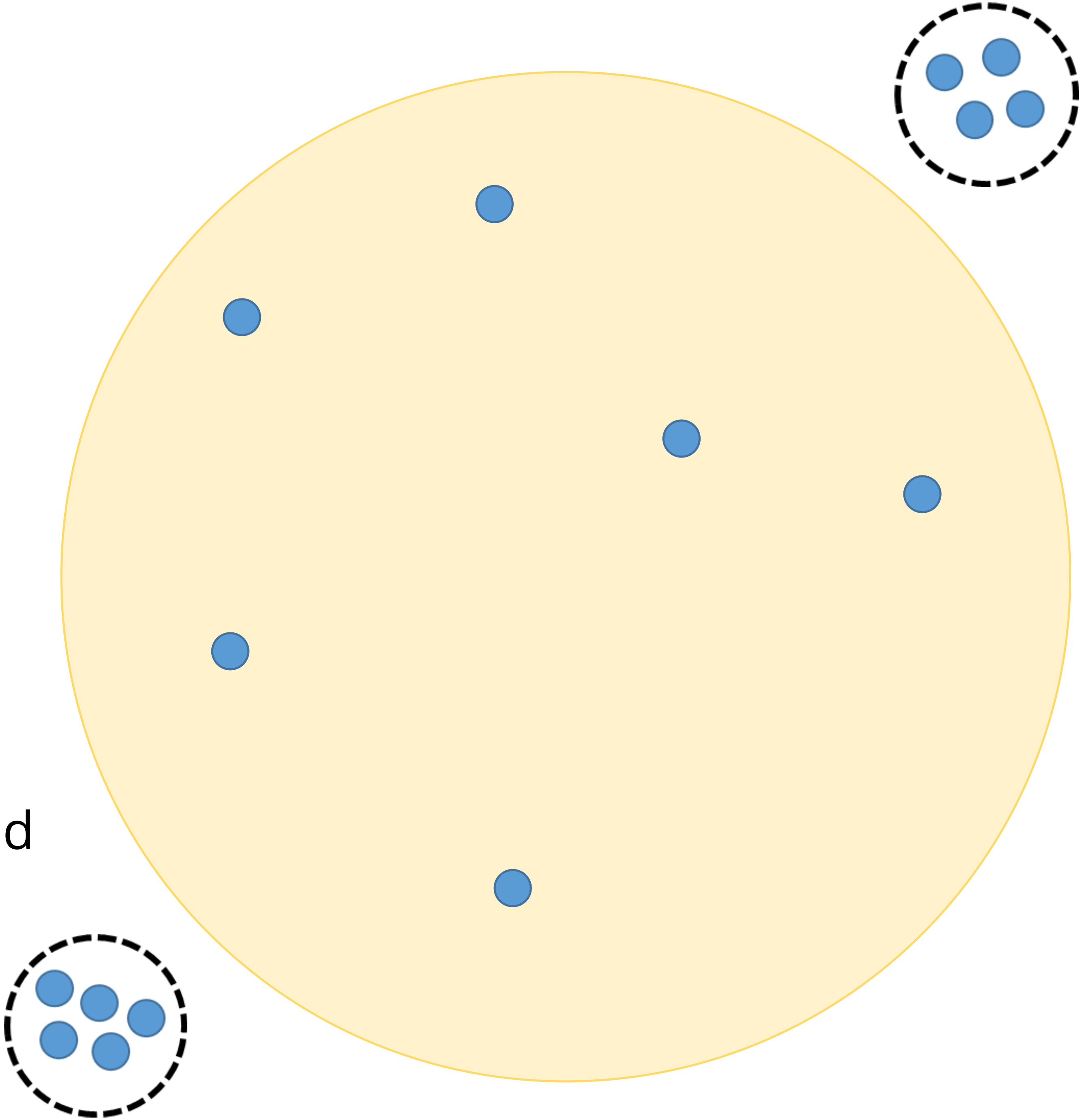
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with **$n^{1-\delta}$** points—until there are none
  - Will handle them separately



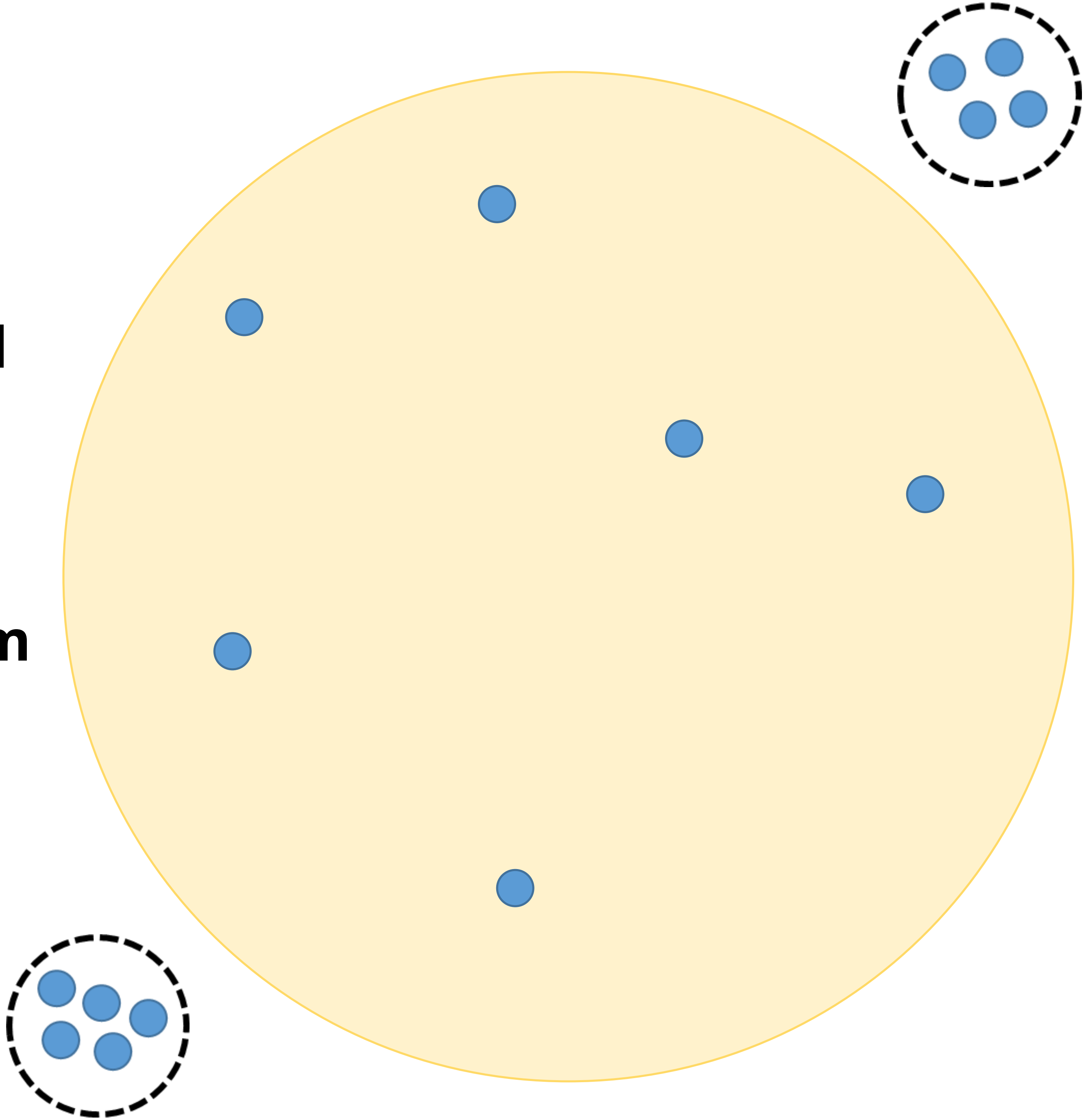
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1-\delta}}$ points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas → points are spread



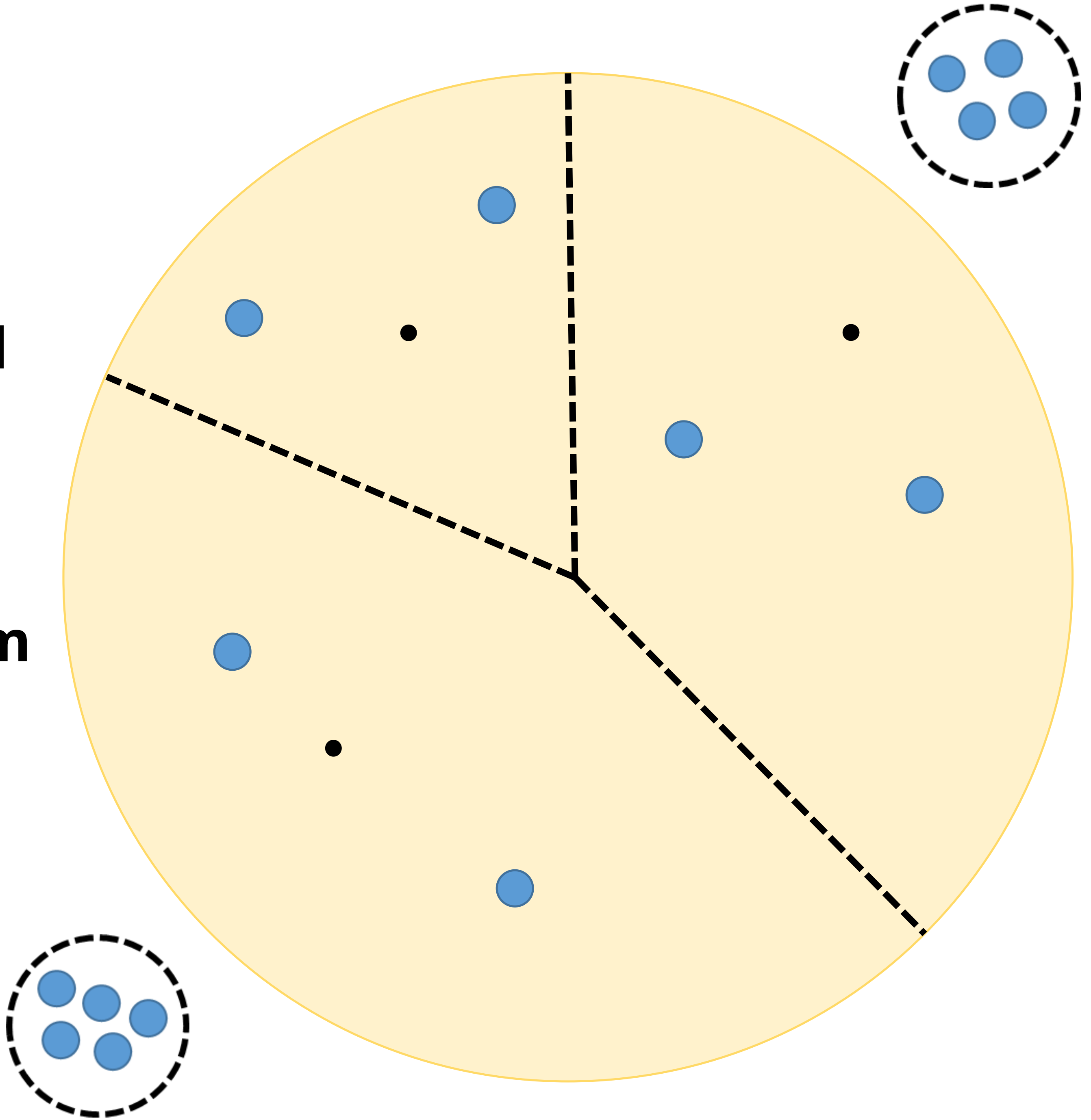
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1-\delta}}$ points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas → points are spread
- Apply Voronoi LSH, recurse
  - dense clusters can appear again!



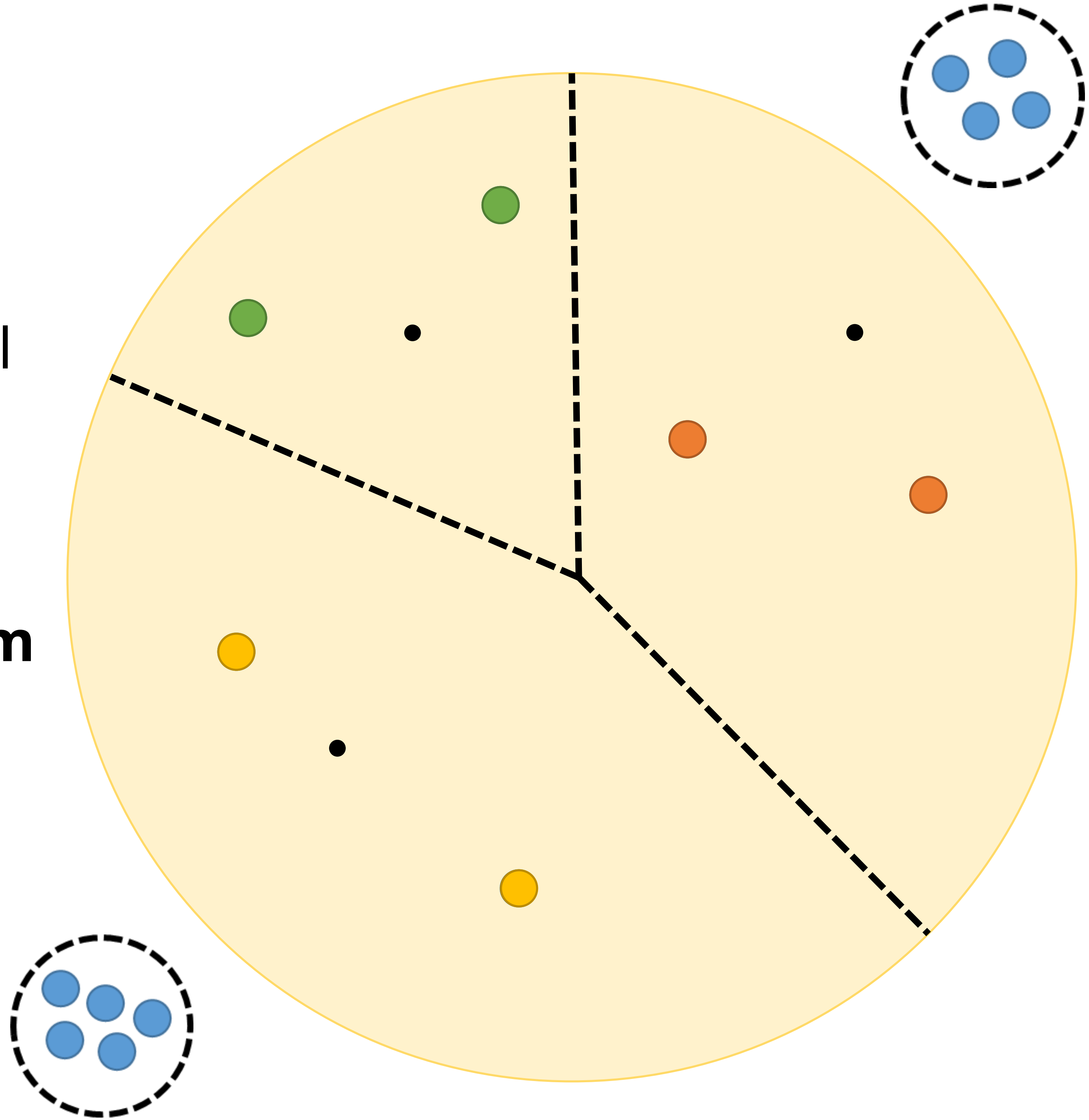
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with $\mathbf{n^{1-\delta}}$ points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas → points are spread
- Apply Voronoi LSH, recurse
  - dense clusters can appear again!



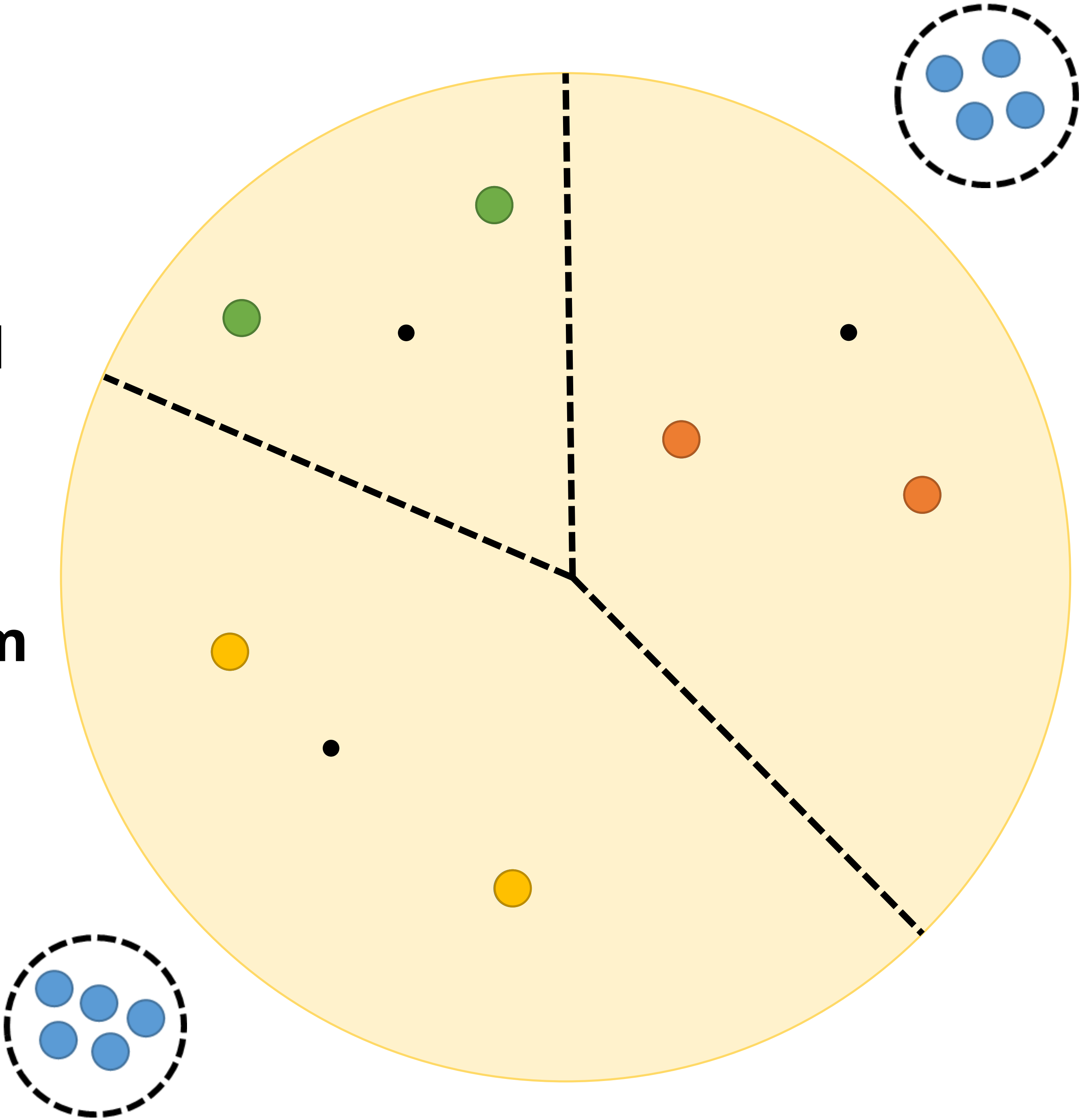
General case

- The dataset does not look random
- Remove *structure*—clusters of small radius with **$n^{1-\delta}$** points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas → points are spread
- Apply Voronoi LSH, recurse
  - dense clusters can appear again!



General case

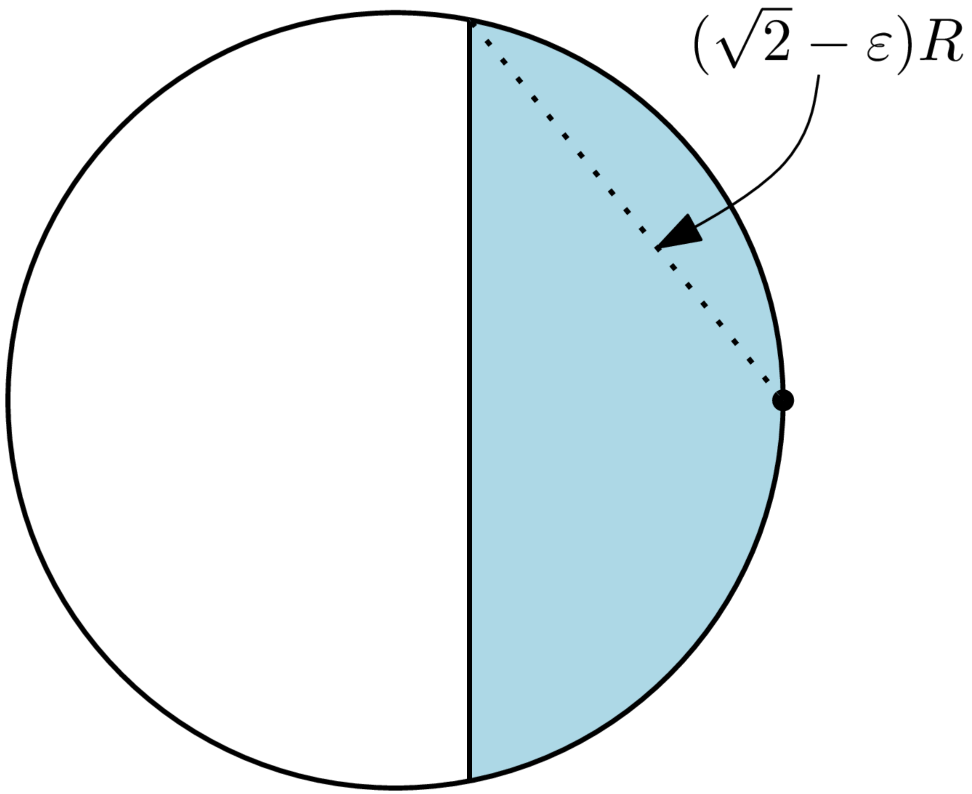
- The dataset does not look random
- Remove *structure*—clusters of small radius with **$n^{1-\delta}$** points—until there are none
  - Will handle them separately
- The remainder **looks like a random set**
  - No dense areas → points are spread
- Apply Voronoi LSH, recurse
  - dense clusters can appear again!
- Query **all** the clusters and **one** part



Handling clusters

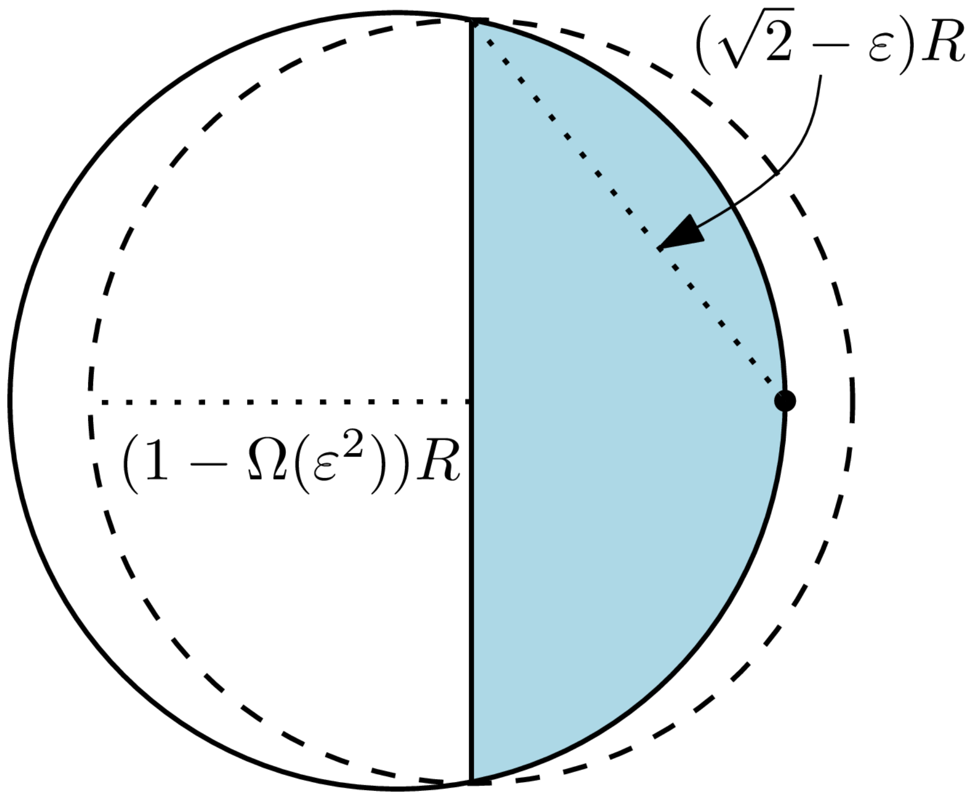
Handling clusters

- Enclose a cluster of radius **$(\sqrt{2} - \varepsilon)R$** in a ball of radius **$(1 - \Omega(\varepsilon^2))\,R$**



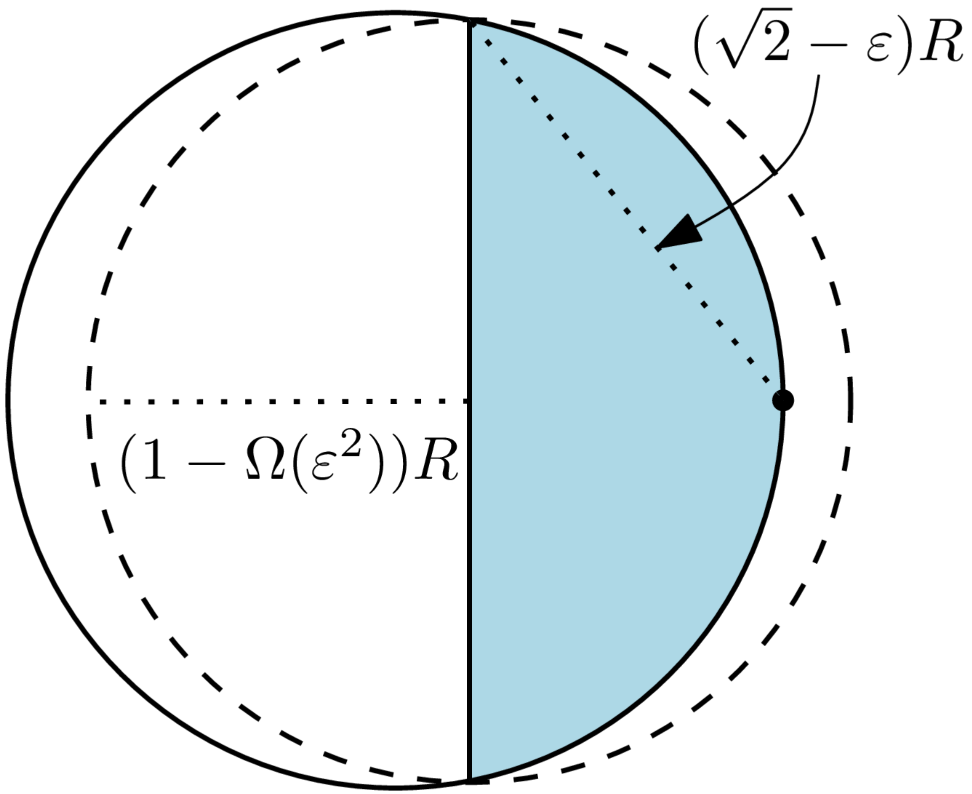
Handling clusters

- Enclose a cluster of radius **$(\sqrt{2} - \varepsilon)R$** in a ball of radius **$(1 - \Omega(\varepsilon^2))\,R$**



Handling clusters

- Enclose a cluster of radius **$(\sqrt{2} - \varepsilon)R$** in a ball of radius **$(1 - \Omega(\varepsilon^2))\,R$**
- Recurse with reduced radius



Overall bookkeeping

Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial

Overall bookkeeping

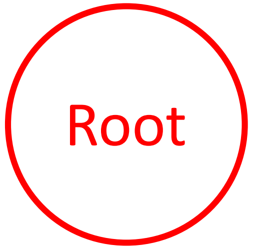
- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well

Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder

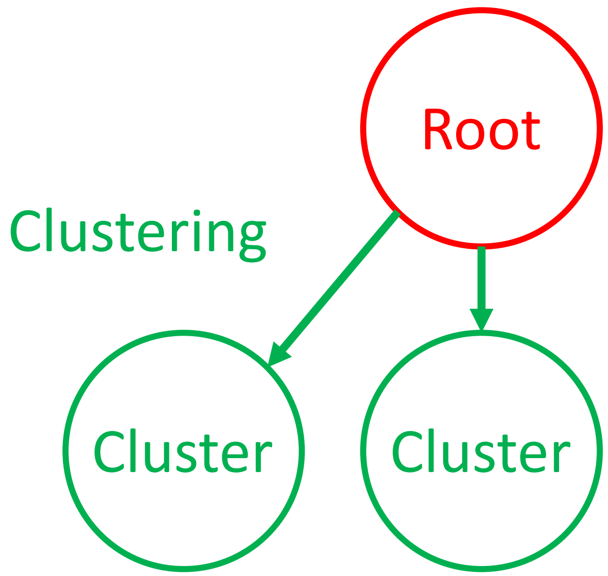
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder



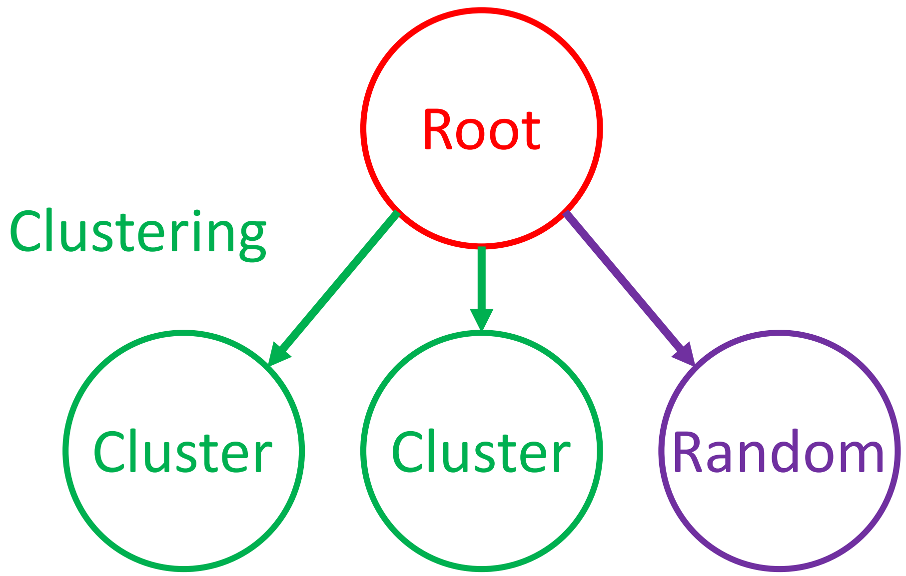
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder



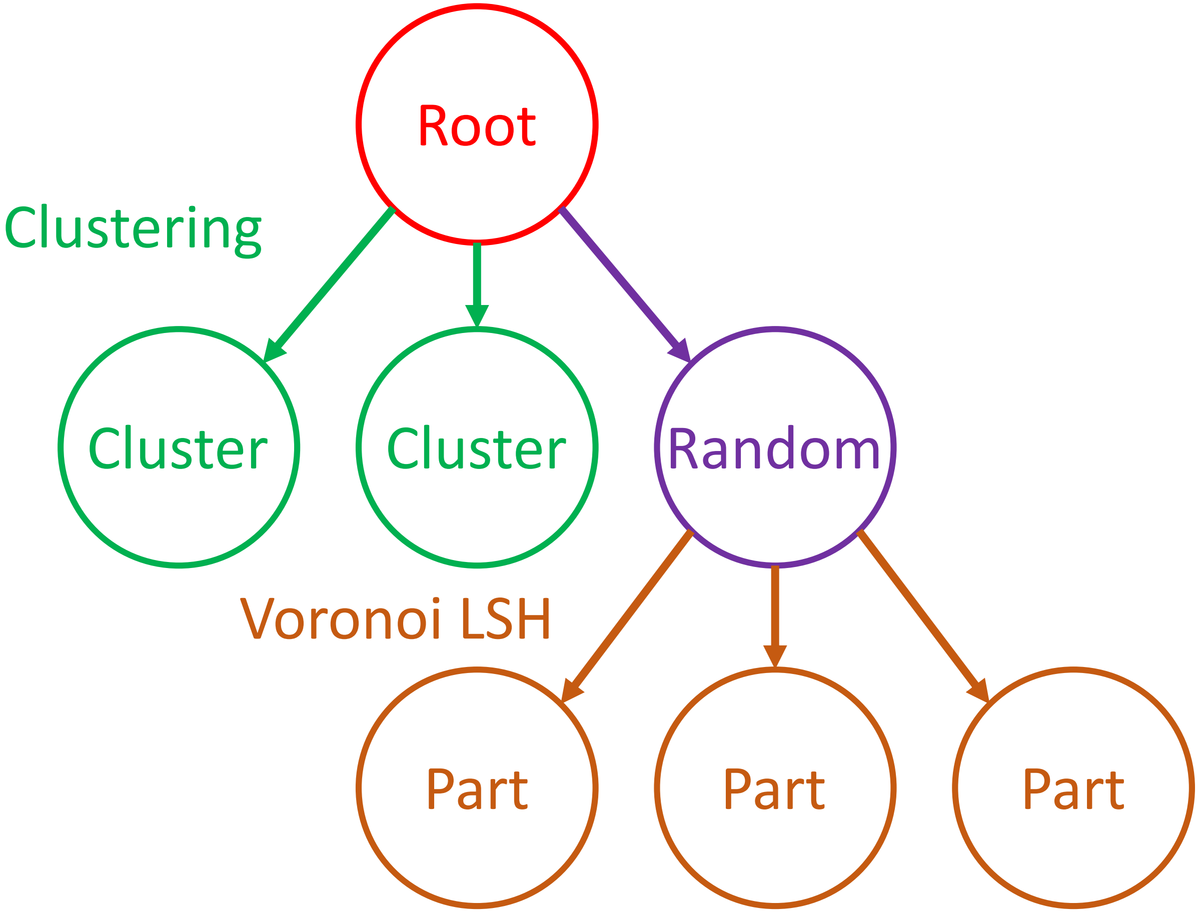
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder



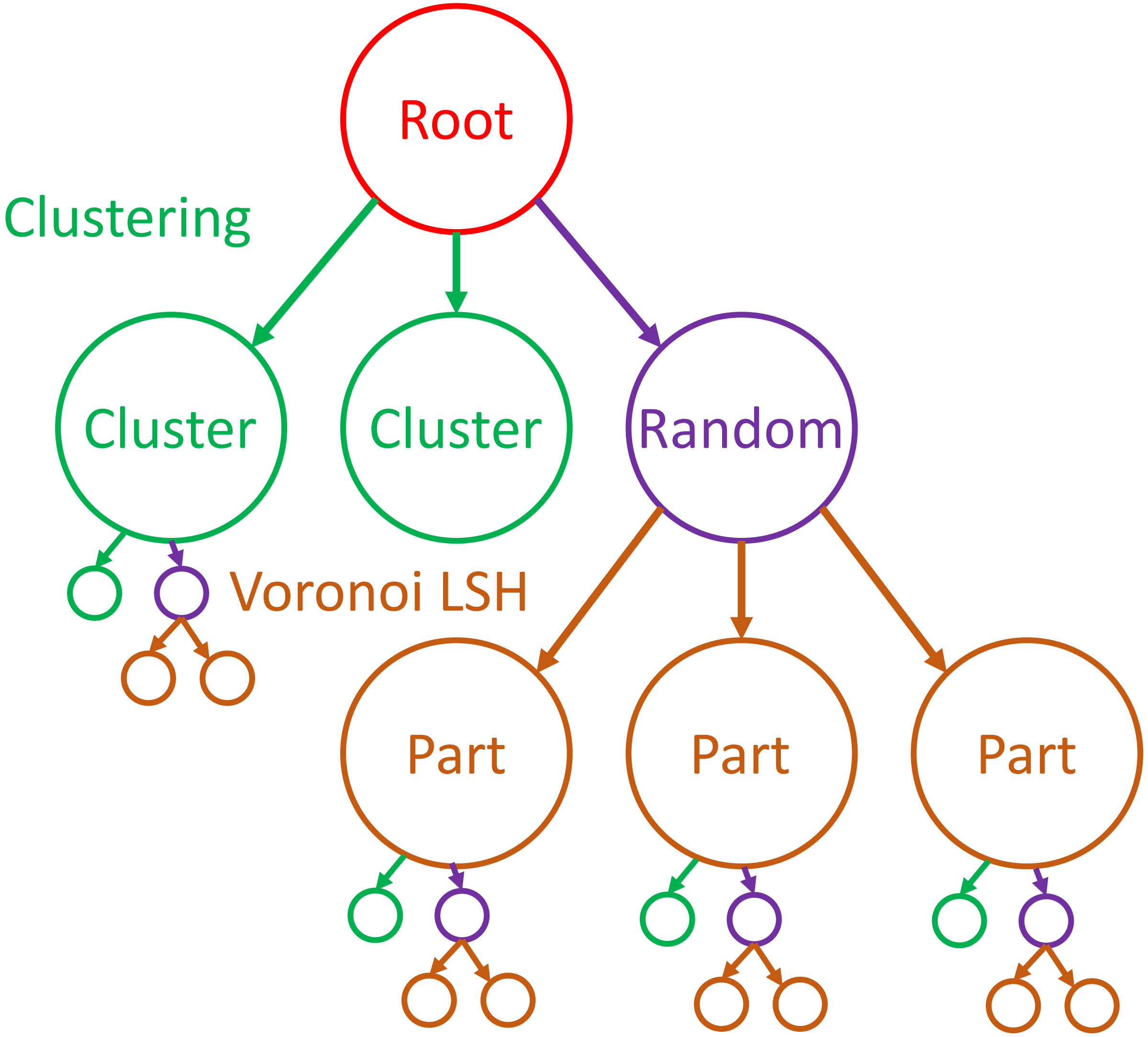
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder



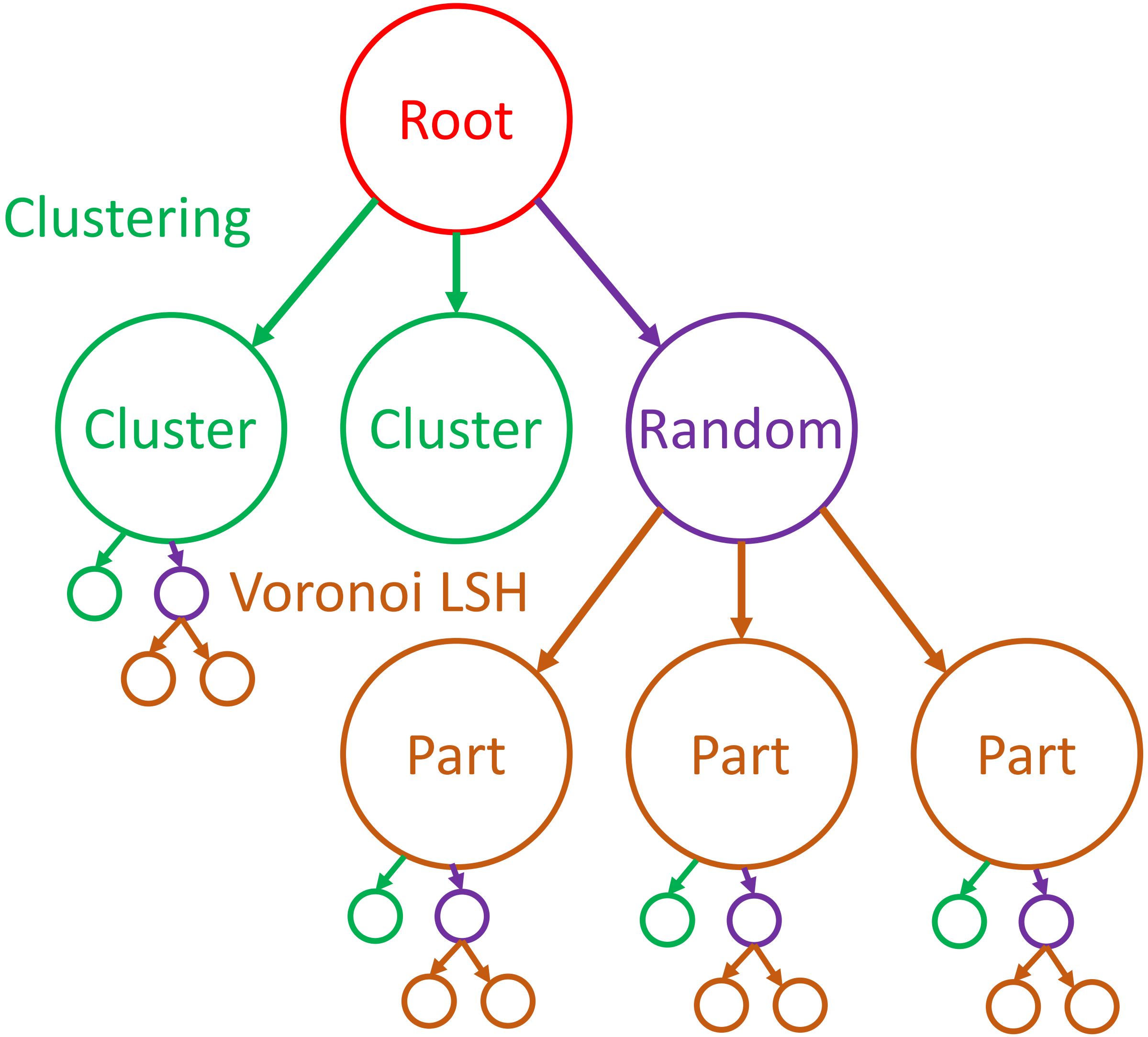
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder



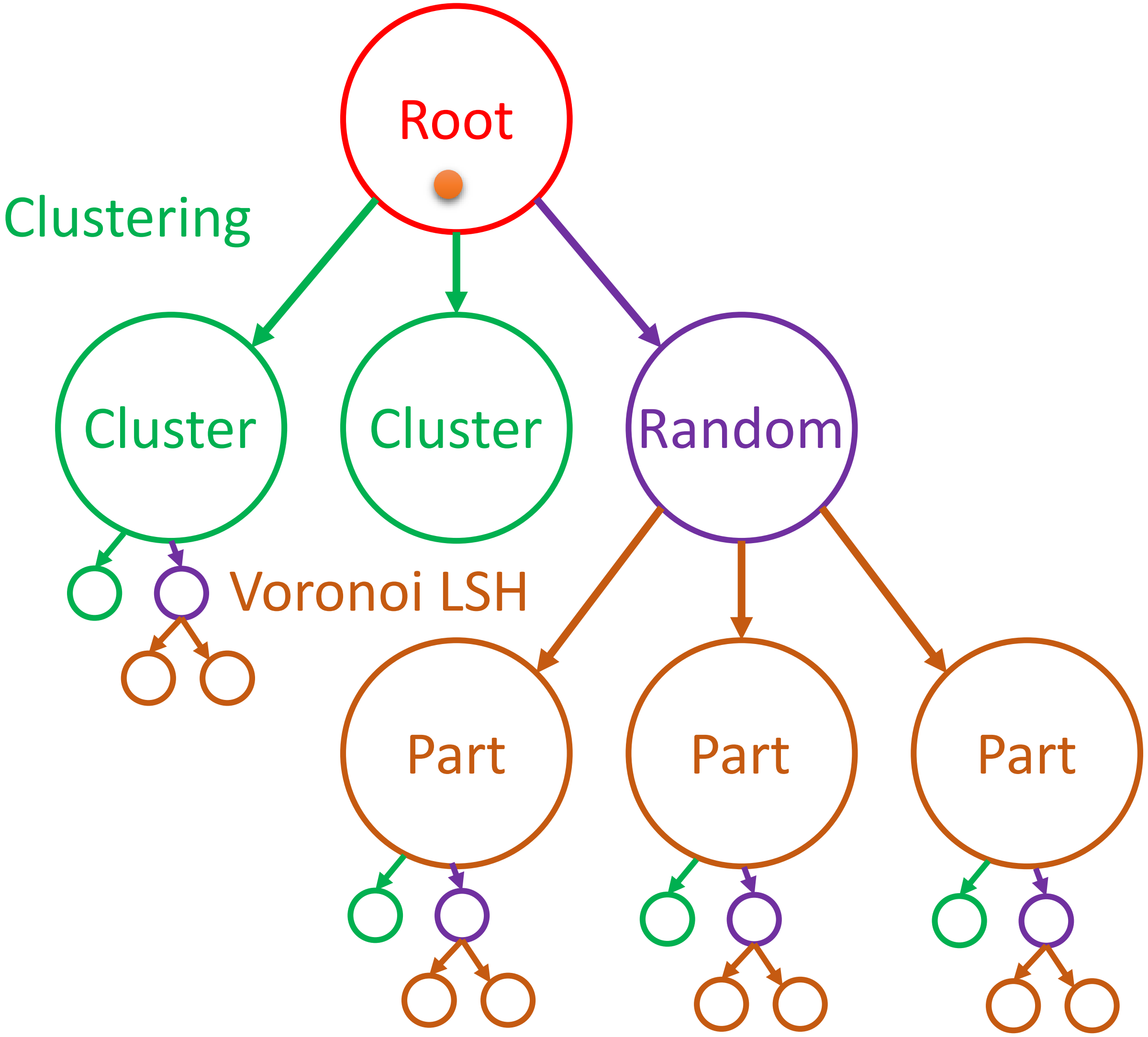
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees



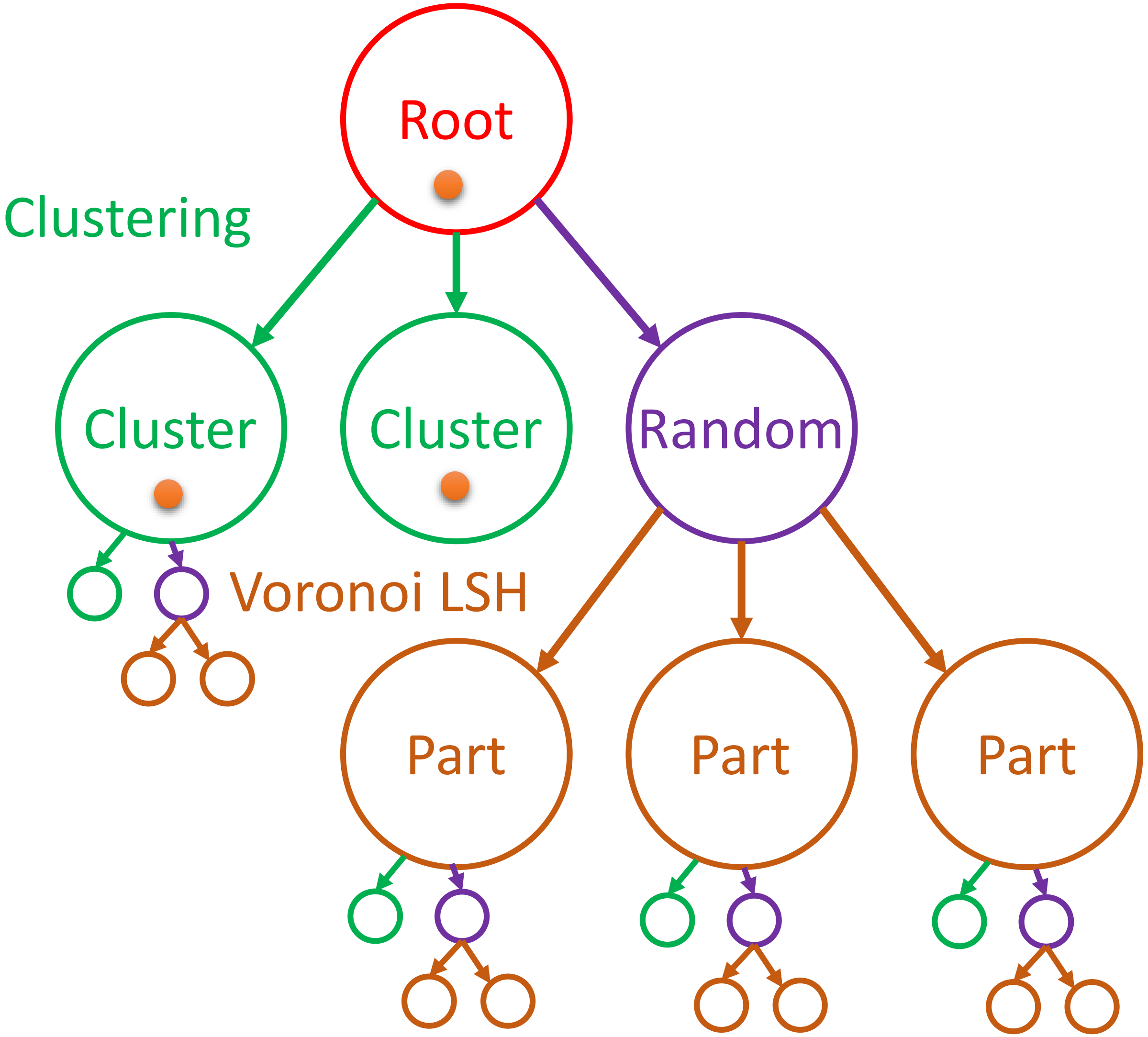
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees



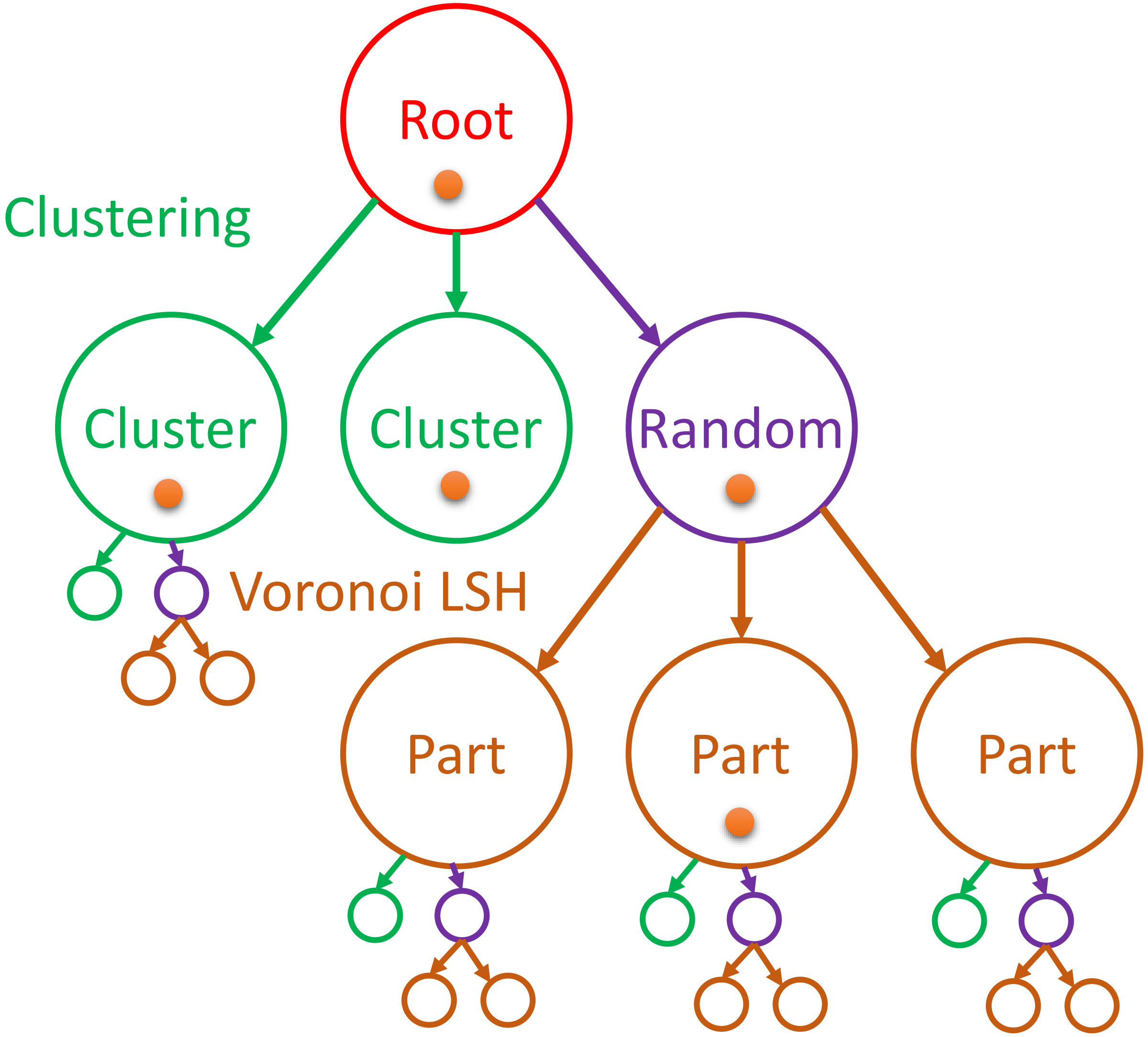
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees



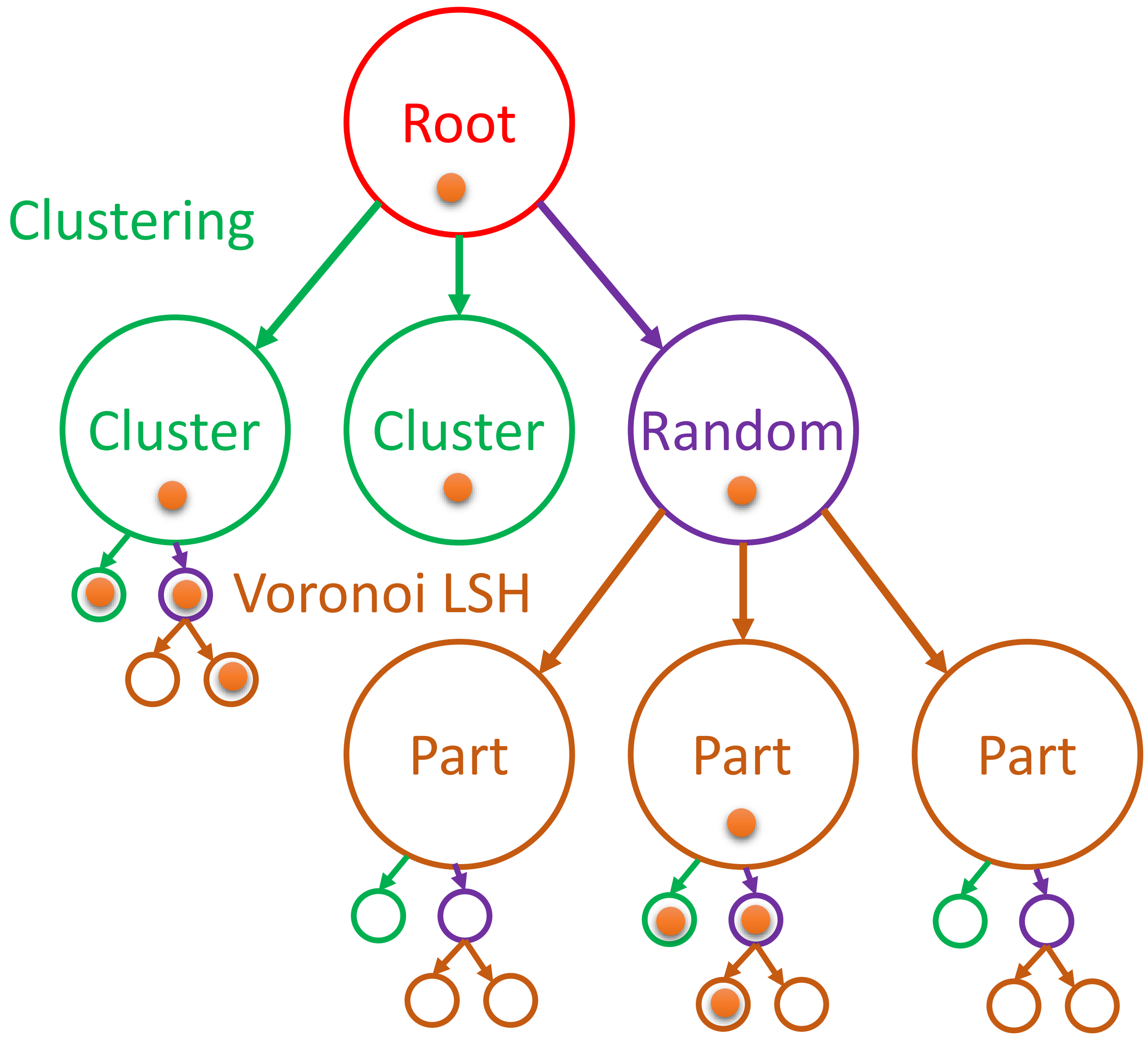
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees



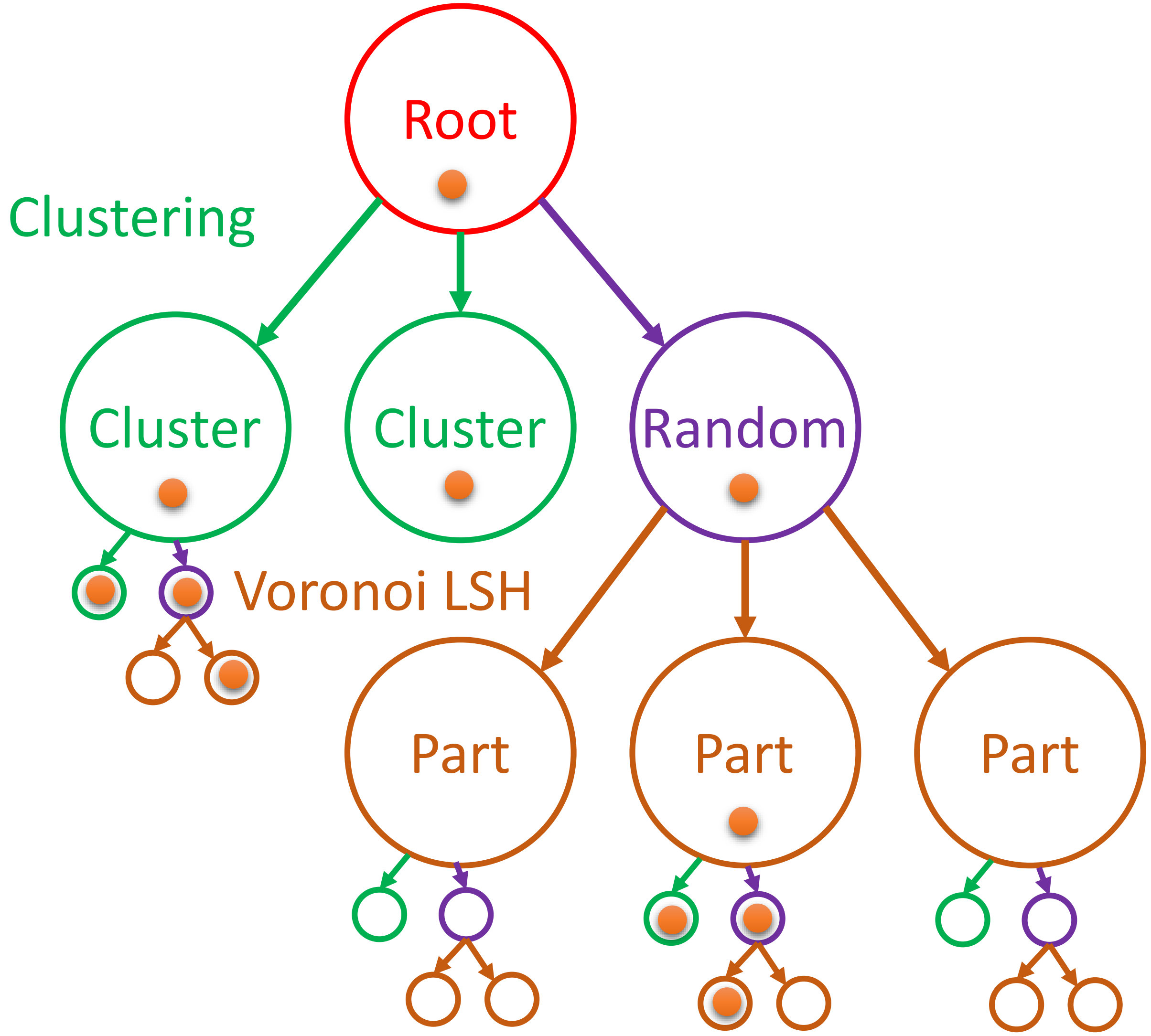
Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees



Overall bookkeeping

- For **clusters** reduce the radius
  - after several reductions the problem becomes trivial
- For the **random remainder**, Voronoi LSH works well
- Can be seen as a decision tree
  - Nodes correspond to clusters and parts of the remainder
  - During the query go to several subtrees
  - A tree occupies space $\mathbf{n^{1+o(1)}}$, query time is $\mathbf{n^{o(1)}}$ (can control depth and branching)
  - Need $\mathbf{n^{\rho}}$ trees to succeed w.h.p.



Outline

Outline

Approximate Near Neighbor Search **(ANN)**

Locality-Sensitive Hashing **(LSH)**

Optimal LSH for a sphere

Beyond LSH for the whole $\mathbf{R}^d$

Practical and optimal LSH for a sphere

Outline

Approximate Near Neighbor Search **(ANN)**

Locality-Sensitive Hashing **(LSH)**

Optimal LSH for a sphere

Beyond LSH for the whole $\mathbf{R}^d$

[Andoni, Indyk, Laarhoven, R, Schmidt 2015] NIPS 2015

Practical and optimal LSH for a sphere

Practicality

Practicality

Is Voronoi LSH practical?

Practicality

Is Voronoi LSH practical?

No!

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement
- Takes **$O(d \cdot T)$** time (even say **$T = 64$** is bad)

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement
- Takes **$O(d \cdot T)$** time (even say **$T = 64$** is bad)

At the same time:

- Hyperplane LSH is *very* useful in practice
- Can practice benefit from theory?

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement
- Takes **$O(d \cdot T)$** time (even say **$T = 64$** is bad)

At the same time:

- Hyperplane LSH is *very* useful in practice
- Can practice benefit from theory?

This work: yes!

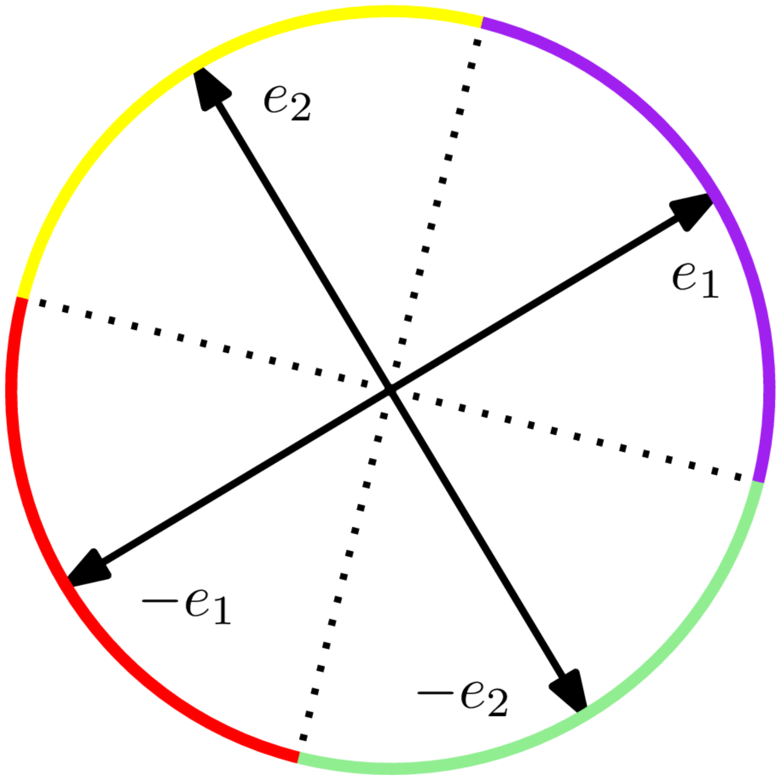
First idea: Cross-polytope LSH

First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope $\{\pm e_i\}$ closest to **Sp**

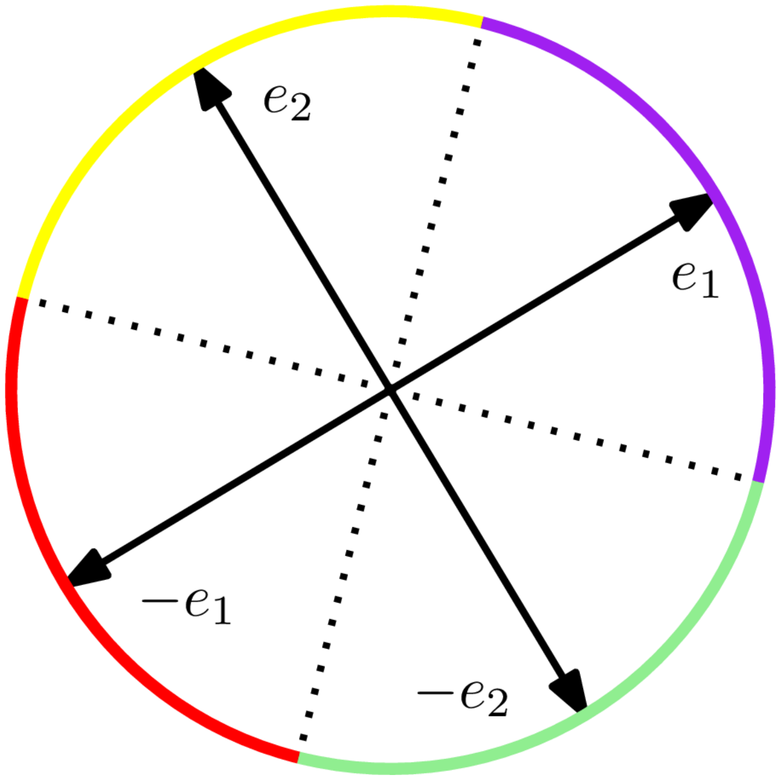
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **{±e$_i$}** closest to **Sp**



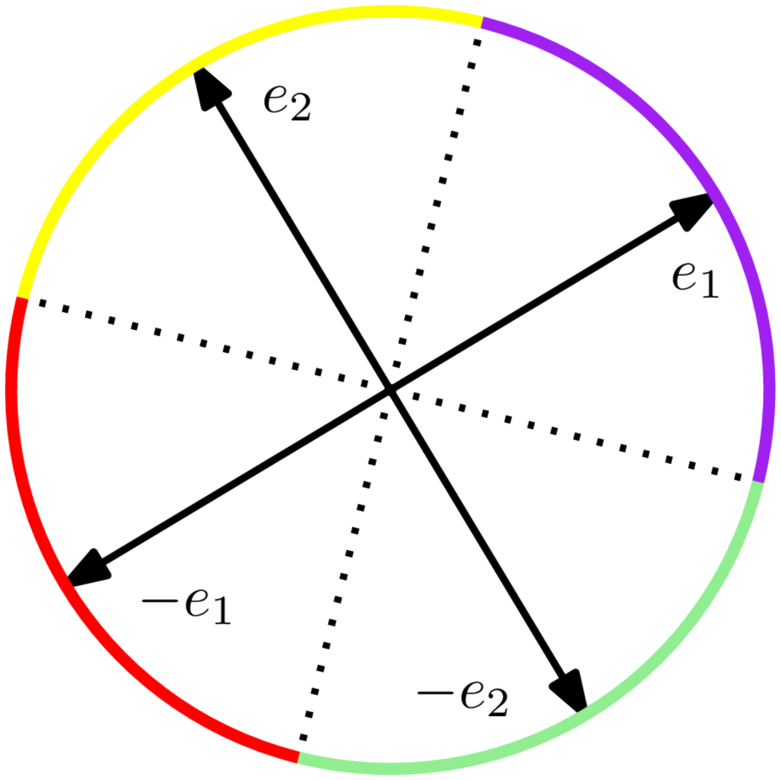
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **{±e$_i$}** closest to **Sp**
- **This paper:** almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!



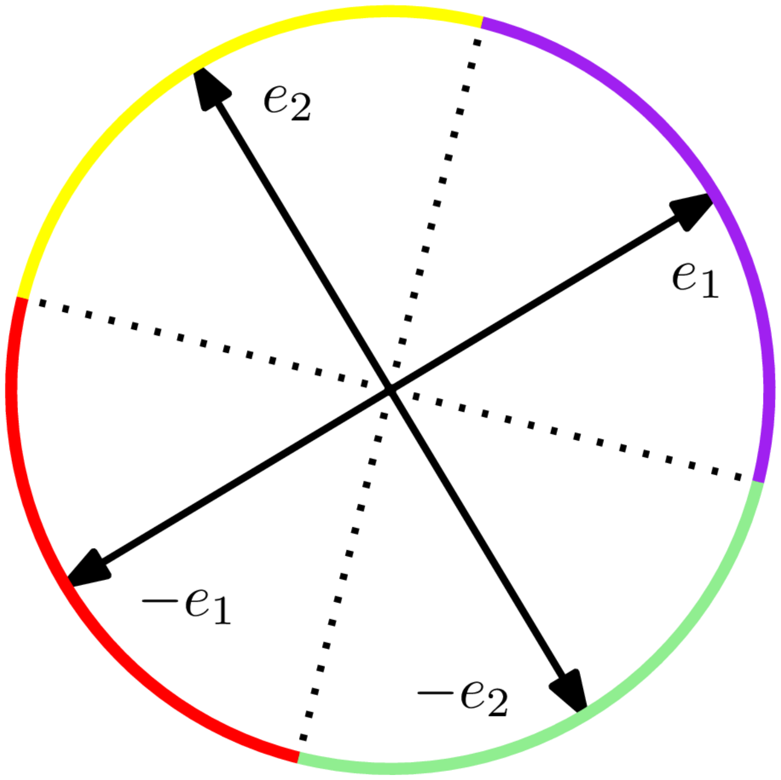
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **$\{\pm e_i\}$** closest to **Sp**
- **This paper:** almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!
- *Impractical:* a random rotation costs **$O(d^2)$** time and space



First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **{±e$_i$}** closest to **Sp**
- **This paper:** almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!
- *Impractical:* a random rotation costs **O(d^2)** time and space
- The second step is cheap (only **O(d)** time)



Second idea: *pseudo*-random rotations

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009],** used in **[Dasgupta, Kumar, Sarlos 2011], [Ailon, Rauhut 2014], [Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009],** used in **[Dasgupta, Kumar, Sarlos 2011], [Ailon, Rauhut 2014], [Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009], **used in **[Dasgupta, Kumar, Sarlos 2011], [Ailon, Rauhut 2014], [Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

$$H_0 = 1$$

$$H_n = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}$$

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

$$\mathbf{p} = (p_1, p_2, \ldots, p_n)$$

↓ **Flip signs**

$$\mathbf{p'} = (\pm p_1, \pm p_2, \ldots, \pm p_n)$$

↓ **Hadamard**

$$\mathbf{Hp'}$$

↓ **Repeat (2-3 times)**

$$H_0 = 1$$

$$H_n = \frac{1}{\sqrt{2}}\begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}$$

Overall hashing scheme

Overall hashing scheme

- Perform 2–3 rounds of “flip signs / Hadamard”

Overall hashing scheme

- Perform 2–3 rounds of “flip signs / Hadamard”
- Find the closest vector from **$\{\pm e_i\}$** (maximum coordinate)

Overall hashing scheme

- Perform 2–3 rounds of “flip signs / Hadamard”
- Find the closest vector from **$\{\pm e_i\}$** (maximum coordinate)
- Evaluation time **$O(d \log d)$**

Overall hashing scheme

- Perform 2–3 rounds of “flip signs / Hadamard”
- Find the closest vector from **$\{\pm e_i\}$** (maximum coordinate)
- Evaluation time **$O(d \log d)$**
- Equivalent to Voronoi LSH with **$T = 2d$** Gaussians

Memory consumption

Memory consumption

- *LSH consumes lots of memory:* myth or reality?

Memory consumption

- *LSH consumes lots of memory:* myth or reality?
- For **$n = 10^6$** random points and queries within **45** degrees, need **725 tables** for success probability **0.9** (if using Hyperplane LSH)

Memory consumption

- *LSH consumes lots of memory:* myth or reality?
- For **$n = 10^6$** random points and queries within **45** degrees, need **725 tables** for success probability **0.9** (if using Hyperplane LSH)
- Can be reduced substantially via **Multiprobe LSH [Lv, Josephson, Wang, Charikar, Li 2007]**

Memory consumption

- *LSH consumes lots of memory:* myth or reality?
- For **$n = 10^6$** random points and queries within **45** degrees, need **725 tables** for success probability **0.9** (if using Hyperplane LSH)
- Can be reduced substantially via **Multiprobe LSH** **[Lv, Josephson, Wang, Charikar, Li 2007]**
- **Our contribution**: Multiprobe for Cross-polytope LSH

Experiments: ANN_SIFT1M

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M, d = 128**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**
- Clustering and re-centering helps
  - Hyperplane: **2.75ms**
  - Cross-polytope: **1.75ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**
- Clustering and re-centering helps
  - Hyperplane: **2.75ms**
  - Cross-polytope: **1.75ms**
- Adding more memory helps

Conclusions and open problems

Conclusions and open problems

- Optimal data-dependent hashing for the whole $\mathbf{L_2}$

Conclusions and open problems

- Optimal data-dependent hashing for the whole $\mathbf{L_2}$
- Practical and optimal LSH for the spherical case

Conclusions and open problems

- Optimal data-dependent hashing for the whole L_2
- Practical and optimal LSH for the spherical case
- Can we make the first bullet practical?
    - Practical “worst-case to random” reduction?

Conclusions and open problems

- Optimal data-dependent hashing for the whole L_2
- Practical and optimal LSH for the spherical case
- Can we make the first bullet practical?
  - Practical “worst-case to random” reduction?

Questions?